OpenFF Evaluator Documentation

openff-evaluator

Nov 13, 2021

GETTING STARTED

An automated and scalable framework for curating, manipulating, and computing data sets of physical properties from molecular simulation and simulation data.

The framework is built around four central ideas:

- **Flexibility:** New physical properties, data sources and calculation approaches are easily added via an extensible plug-in system and a flexible workflow engine.
- **Automation:** *Physical property measurements* are readily importable from open data sources (such as the NIST ThermoML Archive) through the data set APIs, and automatically calculated using either the built-in or user specified calculation schemas.
- **Scalability:** Calculations are readily scalable from single machines and laptops up to large HPC clusters and supercomputers through seamless integration with libraries such as dask.
- **Efficiency:** Properties are estimated using the fastest approach available to the framework, whether that be through evaluating a trained surrogate model, re-evaluating cached simulation data, or by running simulations directly.

CHAPTER

ONE

CALCULATION APPROACHES

The framework is designed around the idea of allowing multiple calculation approaches for estimating the same set of properties, in addition to estimation directly from molecular simulation, all using a uniform API.

The primary purpose of this is to take advantage of the many techniques exist which are able to leverage data from previous simulations to rapidly estimate sets of properties, such as reweighting cached simulation data, or evaluating surrogate models trained upon cached data. The most rapid approach which may accurately estimate a set of properties is automatically determined by the framework on the fly.

Each approach supported by the framework is implemented as a *calculation layer*. Two such layers are currently supported (although new calculation layers can be readily added via the plug-in system):

- evaluating physical properties directly from molecular simulation using the *SimulationLayer*.
- reprocessing cached simulation data with MBAR reweighting using the *ReweightingLayer*.

CHAPTER

TWO

SUPPORTED PHYSICAL PROPERTIES

The framework has built-in support for evaluating a number of *physical properties*, ranging from relatively 'cheap' to compute properties such as liquid densities, up to more computationally demanding properties such as solvation free energies and host-guest binding affinities.

Included for most of these properties is the ability to calculate their derivatives with respect to force field parameters, making the framework ideal for evaluating an objective function and it's gradient as part of a force field optimisation.

Table 1: The physical properties which are natively supported by the framework.

	Direct Simulation		*MBAR Reweighting*	
	Supported	Gradients	Supported	Gradients
Density	✓	✓	✓	✓
Dielectric Constant	✓	✓*	✓	✓*
$H_{\text{vaporization}}$	✓	✓	✓	✓
H_{mixing}	✓	✓	✓*	✓
V_{excess}	✓	✓	✓	✓
$G_{\text{solvation}}$	✓	✓*	×	×
$G_{\text{host-guest}}$ **(beta)**	✓*	×	×	×

** Entries marked with an asterisk are supported but have not yet been extensively tested and validated.*

See the *physical properties overview page* for more details.

2.1 Installation

The OpenFF Evaluator is currently installable either through `conda` or directly from the source code. Whichever route is chosen, it is recommended to install the framework within a conda environment and allow the conda package manager to install the required and optional 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.

2.1.1 Installation from Conda

To install the `openff-evaluator` from the `conda-forge` channel simply run:

```
conda install -c conda-forge openff-evaluator
```

2.1.2 Recommended Dependencies

If you have access to the fantastic OpenEye toolkit we recommend installing this to enable (among many other things) the use of the `BuildDockedCoordinates` protocol and faster conformer generation / AM1BCC partial charge calculations:

```
conda install -c openeye openeye-toolkits
```

To parameterize systems with the Amber `tleap` tool using a `TLeapForceFieldSource` the `ambertools` package must be installed:

```
conda install -c conda-forge 'ambertools >=19.0'
```

2.1.3 Installation from Source

To install the OpenFF Evaluator from source begin by cloning the repository from github:

```
git clone https://github.com/openforcefield/openff-evaluator.git
cd openff-evaluator
```

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

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

Finally, install the estimator itself:

```
python setup.py develop
```

2.2 Architecture

The openff-evaluator framework is constructed as a collection of modular components, each performing a specific role within the estimation of physical property data sets. These components are designed to be as extensible as possible, with support for user created plug-ins built into their core.

Fig. 1: An overview of the openff-evaluators modular design. The framework is split into a 'client-side' which handles the curation and preparation of data sets, and a 'server-side' which performs the estimation of the data sets.

The framework is implemented as a *client-server* architecture. This design allows users to spin up *Evaluator Server* instances on whichever compute resources they may have available (from a single machine up to a large HPC cluster), and to which *Evaluator Client* objects may connect to both request that data sets be estimated, and to query and retrieve the results of those requests.

The *client-side* of the framework is predominantly responsible for providing APIs and objects for:

- curating *data sets* of physical properties from open data sources.
- specifing custom *calculation schemas* which describe how individual properties should be computed.
- requesting that data sets be estimated by a running *Evaluator Server* instance.
- retrieving the results of estimation requests from a running *Evaluator Server* instance.

while the *server-side* is responsible for:

- receiving estimation requests from an *Evaluator Client* object.
- automatically determining which *calculation approach* to use for each property in the request.
- executing those requests across the available *compute resources* following the calculation schemas provided by the client
- *caching data* from any calculations which may be useful for future calculations.

All communication between servers and clients is handled through the TCP protocol.

2.3 Evaluator Client

The `EvaluatorClient` object is responsible for both submitting requests to estimate a data set of properties to a running *Evaluator Server* instance, and for pulling back the results of those requests when complete.

An `EvaluatorClient` object may optionally be created using a set of `ConnectionOptions` which specifies the network address of the running *Evaluator Server* instance to connect to:

```
# Specify the address of a server running on the local machine.
connection_options = ConnectionOptions(server_address="localhost", server_port=8000)
# Create the client object
evaluator_client = EvaluatorClient(connection_options)
```

2.3.1 Requesting Estimates

The client can request the estimation of a data set of properties using the *request_estimate()* function:

```
# Specify the data set.
data_set = PhysicalPropertyDataSet()
data_set.add_properties(...)

# Specify the force field source.
force_field = SmirnoffForceFieldSource.from_path("openff-1.0.0.offxml")

# Specify some estimation options (optional).
options = client.default_request_options(data_set, force_field)

# Specify the parameters to differentiate with respect to (optional).
gradient_keys = [
    ParameterGradientKey(tag="vdW", smirks="[#6X4:1]", attribute="epsilon")
]

# Request the estimation of the data set.
request, errors = evaluator_client.request_estimate(
    data_set,
    force_field,
    options,
    gradient_keys
)
```

A request must at minimum specify:

- the *data set* of physical properties to estimate.
- the *force field parameters* to estimate the data set using.

and may also optionally specify:

- the *options* to use when estimating the property set.
- the parameters to differentiate each physical property estimate with respect to.

Note: Gradients can currently only be computed for requests using a SMIRNOFF based force field.

The *request_estimate()* function returns back two objects:

- a *Request* object which can be used to retrieve the results of the request and,
- an *EvaluatorException* object which will be populated if any errors occured while submitting the request.

The *Request* object is similar to a Future object, in that it is an object which can be used to query the current status of a request either asynchronously:

```
results = request.results(synchronous=False)
```

or synchronously:

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

The results (which may currently be incomplete) are returned back as a *RequestResult* object.

The `Request` object is fully JSON serializable:

```
# Save the request to JSON
request.json(file_path="request.json", format=True)
# Load the request from JSON
request = Request.from_json(file_path="request.json")
```

making it easy to keep track of any open requests.

2.3.2 Request Options

The `RequestOptions` object allows greater control over how properties are estimated by the server. It currently allows control over:

- `calculation_layers`: The *calculation layers* which the server should attempt to use when estimating the data set. The order which the layers are specified in this list is the order which the server will attempt to use each layer.
- `calculation_schemas`: The *calculation schemas* to use for each allowed calculation layer per class of property. These will be automatically populated in the cases where no user specified schema is provided, and where a default schema has been registered with the plugin system for the particular layer and property type.

If no options are passed to `request_estimate()` a default set will be generated through a call to `default_request_options()`. For more information about how default calculation schemas are registered, see the *Default Schemas* section.

2.3.3 Force Field Sources

Different force field representations (e.g. `SMIRNOFF`, `TLeap`, `LigParGen`) are defined within the framework as `ForceFieldSource` objects. A force field source should specify *all* of the options which would be required by a particular force field, such as the non-bonded cutoff or the charge scheme if not specified directly in the force field itself.

Currently the framework has built in support for force fields applied via:

- the OpenFF toolkit (`SmirnoffForceFieldSource`).
- the `tleap` program from the AmberTools suite (`LigParGenForceFieldSource`).
- an instance of the LigParGen server (`LigParGenForceFieldSource`).

The client will automatically adapt any of the built-in calculation schemas which are based off of the `WorkflowCalculationSchema` to use the correct workflow protocol (`BuildSmirnoffSystem`, `BuildTLeapSystem` or `BuildLigParGenSystem`) for the requested force field.

2.4 Evaluator Server

The `EvaluatorServer` object is responsible for coordinating the estimation of physical property data sets as requested by *evaluator clients*. Its primary responsibilities are to:

- recieve incoming requests from an *evaluator clients* to either estimate a dataset of properties, or to query the status of a previous request.
- request that each specified *calculation layers* attempt to estimate the data set of properties, cascading unestimated properties through the different layers.

An `EvaluatorServer` must be created with an accompanying *calculation backend* which will be responsible for distributing any calculations launched by the different calculation layers:

```
with DaskLocalCluster() as calculation_backend:

    evaluator_server = EvaluatorServer(calculation_backend)
    evaluator_server.start()
```

It may also be optionally created using a specific *storage backend* if the default `LocalFileStorage` is not sufficient:

```
with DaskLocalCluster() as calculation_backend:

    storage_backend = LocalFileStorage()

    evaluator_server = EvaluatorServer(calculation_backend, storage_backend)
    evaluator_server.start()
```

By default the server will run synchronously until it is killed, however it may also be run asynchronously such that it can be interacted with directly by a client in the same script:

```
with DaskLocalCluster() as calculation_backend:

    with EvaluatorServer(calculation_backend) as evaluator_server:

        # Specify the data set.
        data_set = PhysicalPropertyDataSet()
        data_set.add_properties(...)

        # Specify the force field source.
        force_field = SmirnoffForceFieldSource.from_path("openff-1.0.0.offxml")

        # Request the estimation of the data set.
        request, errors = evaluator_client.request_estimate(data_set,force_field)
        # Wait for the results.
        results = request.results(synchronous=True)
```

2.4.1 Estimation Batches

When a server recieves a request from a client, it will attempt to split the requested set of properties into smaller batches, represented by the `Batch` object. The server is currently only able to mark entire batches of estimated properties as being completed, as opposed to individual properties.

Currently the server supports two ways of batching properties:

- `SameComponents`: All properties measured for the substance containing the *same* components will be batched together. As an example, the density of a 80:20 and a 20:80 mix of ethanol and water would be batched together, but the density of pure ethanol and the density of pure water would be placed into separate batches.
- `SharedComponents`: All properties measured for substances containing at least one common component will be batched together. As an example, the densities of 80:20 and 20:80 mixtures of ethanol and water, and the pure densities of ethanol and water would be batched together.

The mode of batching is set by the client using the `batch_mode` attribute of the request options.

2.5 Tutorial 01 - Loading Data Sets

In this tutorial we will be exploring the frameworks utilities for loading and manipulating data sets of physical property measurements. The tutorial will cover

- Loading a data set of density measurements from NISTs ThermoML Archive
- Filtering the data set down using a range of criteria, including temperature pressure, and composition.
- Supplementing the data set with enthalpy of vaporization (ΔH_v) data sourced directly from the literature

If you haven't yet installed the OpenFF Evaluator framework on your machine, check out the *installation instructions here!*

Note: If you are running this tutorial in google colab you will need to run a setup script instead of following the installation instructions:

```
[1]: # !wget https://raw.githubusercontent.com/openforcefield/openff-evaluator/master/docs/
     ↪tutorials/colab_setup.ipynb
     # %run colab_setup.ipynb
```

For the sake of clarity all warnings will be disabled in this tutorial:

```
[2]: import warnings
     warnings.filterwarnings('ignore')
     import logging
     logging.getLogger("openff.toolkit").setLevel(logging.ERROR)
```

2.5.1 Extracting Data from ThermoML

For anyone who is not familiar with the ThermoML archive - it is a fantastic database of physical property measurements which have been extracted from data published in the

- Journal of Chemical and Engineering Data
- Journal of Chemical Thermodynamics
- Fluid Phase Equilibria
- Thermochimica Acta
- International Journal of Thermophysics

journals. It includes data for a wealth of different physical properties, from simple densities and melting points, to activity coefficients and osmotic coefficients, all of which is freely available. As such, it serves as a fantastic resource for benchmarking and optimising molecular force fields against.

The Evaluator framework has built-in support for extracting this wealth of data, storing the data in easy to manipulate python objects, and for automatically re-computing those properties using an array of calculation techniques, such as molecular simulations and, in future, from trained surrogate models.

This support is provided by the `ThermoMLDataSet` object:

```
[3]: from openff.evaluator.datasets.thermoml import ThermoMLDataSet
```

The `ThermoMLDataSet` object offers two main routes for extracting data the the archive:

- extracting data directly from the NIST ThermoML web server

- extracting data from a local ThermoML XML archive file

Here we will be extracting data directly from the web server. To pull data from the web server we need to specifiy the digital object identifiers (DOIs) of the data we wish to extract - these correspond to the DOI of the publication that the data was initially sourced from.

For this tutorial we will be extracting data using the following DOIs:

[4]:
```
data_set = ThermoMLDataSet.from_doi(
    "10.1016/j.fluid.2013.10.034",
    "10.1021/je1013476",
)
```

We can inspect the data set to see how many properties were loaded:

[5]:
```
len(data_set)
```

[5]:
```
275
```

and for how many different substances those properties were measured for:

[6]:
```
len(data_set.substances)
```

[6]:
```
254
```

We can also easily check which types of properties were loaded in:

[7]:
```
print(data_set.property_types)
```

```
{'EnthalpyOfMixing', 'Density'}
```

2.5.2 Filtering the Data Set

The data set object we just created contains many different functions which will allow us to filter the data down, retaining only those measurements which are of interest to us.

The first thing we will do is filter out all of the measurements which aren't density measurements:

[8]:
```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByPropertyTypes,
    FilterByPropertyTypesSchema
)

data_set = FilterByPropertyTypes.apply(
    data_set, FilterByPropertyTypesSchema(property_types=["Density"])
)

print(data_set.property_types)
```

```
{'Density'}
```

Next we will filter out all measurements which were made away from atmospheric conditions:

[9]:
```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByPressure,
    FilterByPressureSchema,
```

(continues on next page)

(continued from previous page)

```
    FilterByTemperature,
    FilterByTemperatureSchema,
)

print(f"There were {len(data_set)} properties before filtering")

# First filter by temperature.
data_set = FilterByTemperature.apply(
    data_set,
    FilterByTemperatureSchema(minimum_temperature=298.0, maximum_temperature=298.2)
)
# and then by pressure
data_set = FilterByPressure.apply(
    data_set,
    FilterByPressureSchema(minimum_pressure=101.224, maximum_pressure=101.426)
)

print(f"There are now {len(data_set)} properties after filtering")
```

```
There were 213 properties before filtering
There are now 9 properties after filtering
```

Finally, we will filter out all measurements which were not measured for either ethanol (CCO) or isopropanol (CC(C)O):

[10]:
```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterBySmiles,
    FilterBySmilesSchema,
)

data_set = FilterBySmiles.apply(
    data_set,
    FilterBySmilesSchema(smiles_to_include=["CCO", "CC(C)O"])
)

print(f"There are now {len(data_set)} properties after filtering")
```

```
There are now 2 properties after filtering
```

We will convert the filtered data to a pandas `DataFrame` to more easily visualize the final data set:

[11]:
```
pandas_data_set = data_set.to_pandas()
pandas_data_set[
    ["Temperature (K)", "Pressure (kPa)", "Component 1", "Density Value (g / ml)",
↪"Source"]
].head()
```

[11]:
```
   Temperature (K)  Pressure (kPa) Component 1  Density Value (g / ml)  \
0           298.15         101.325      CC(C)O                 0.78270
1           298.15         101.325         CCO                 0.78507

                     Source
0  10.1016/j.fluid.2013.10.034
1            10.1021/je1013476
```

Through filtering, we have now cut down from over 250 property measurements down to just 2. There are many more possible filters which can be applied. All of these and more information about the data set object can be found in the `PhysicalPropertyDataSet` (from which the `ThermoMLDataSet` class inherits) API documentation.

2.5.3 Adding Extra Data

For the final part of this tutorial, we will be supplementing our newly filtered data set with some enthalpy of vaporization (ΔH_v) measurements sourced directly from the literature (as opposed to from the ThermoML archive).

We will be sourcing values of the ΔH_v of ethanol and isopropanol, summarised in the table below, from the Enthalpies of vaporization of some aliphatic alcohols publication:

Compound	Temperature / K	ΔH_v / $kJmol^{-1}$	$\delta\Delta H_v$ / $kJmol^{-1}$
Ethanol	298.15	42.26	0.02
Isopropanol	298.15	45.34	0.02

In order to create a new ΔH_v measurements, we will first define the state (namely temperature and pressure) that the measurements were recorded at:

[12]:
```
from openff.evaluator import unit
from openff.evaluator.thermodynamics import ThermodynamicState

thermodynamic_state = ThermodynamicState(
    temperature=298.15 * unit.kelvin, pressure=1.0 * unit.atmosphere
)
```

Note: Here we have made use of the ``openff.evaluator.unit`` module to attach units to the temperatures and pressures we are filtering by. This module simply exposes a ``UnitRegistry`` from the fantasticpintlibrary. Pint provides full support for attaching to units to values and is used extensively throughout this framework.

the substances that the measurements were recorded for:

[13]:
```
from openff.evaluator.substances import Substance

ethanol = Substance.from_components("CCO")
isopropanol = Substance.from_components("CC(C)O")
```

and the source of this measurement (defined as the DOI of the publication):

[14]:
```
from openff.evaluator.datasets import MeasurementSource

source = MeasurementSource(doi="10.1016/S0021-9614(71)80108-8")
```

We will combine this information with the values of the measurements to create an object which encodes each of the ΔH_v measurements

[15]:
```
from openff.evaluator.datasets import PropertyPhase
from openff.evaluator.properties import EnthalpyOfVaporization

ethanol_hvap = EnthalpyOfVaporization(
    thermodynamic_state=thermodynamic_state,
    phase=PropertyPhase.Liquid | PropertyPhase.Gas,
    substance=ethanol,
    value=42.26*unit.kilojoule / unit.mole,
```

(continues on next page)

(continued from previous page)

```
    uncertainty=0.02*unit.kilojoule / unit.mole,
    source=source
)
isopropanol_hvap = EnthalpyOfVaporization(
    thermodynamic_state=thermodynamic_state,
    phase=PropertyPhase.Liquid | PropertyPhase.Gas,
    substance=isopropanol,
    value=45.34*unit.kilojoule / unit.mole,
    uncertainty=0.02*unit.kilojoule / unit.mole,
    source=source
)
```

These properties can then be added to our data set:

[16]:
```
data_set.add_properties(ethanol_hvap, isopropanol_hvap)
```

If we print the data set again using pandas we should see that our new measurements have been added:

[17]:
```
pandas_data_set = data_set.to_pandas()
pandas_data_set[
    ["Temperature (K)",
     "Pressure (kPa)",
     "Component 1",
     "Density Value (g / ml)",
     "EnthalpyOfVaporization Value (kJ / mol)",
     "Source"
     ]
].head()
```

[17]:
```
   Temperature (K)  Pressure (kPa) Component 1  Density Value (g / ml)  \
0           298.15         101.325      CC(C)O                 0.78270
1           298.15         101.325         CCO                 0.78507
2           298.15         101.325         CCO                     NaN
3           298.15         101.325      CC(C)O                     NaN

   EnthalpyOfVaporization Value (kJ / mol)                        Source
0                                      NaN    10.1016/j.fluid.2013.10.034
1                                      NaN              10.1021/je1013476
2                                    42.26  10.1016/S0021-9614(71)80108-8
3                                    45.34  10.1016/S0021-9614(71)80108-8
```

2.5.4 Conclusion

We will finish off this tutorial by saving the data set we have created as a JSON file for future use:

[18]:
```
data_set.json("filtered_data_set.json", format=True);
```

And that concludes the first tutorial. For more information about data sets in the Evaluator framework check out the *data set* and *ThermoML* documentation.

In the next tutorial we will be estimating the data set we have created here using molecular simulation.

If you have any questions and / or feedback, please open an issue on the GitHub issue tracker.

2.6 Tutorial 02 - Estimating Data Sets

In this tutorial we will be estimating the data set we created in the *first tutorial* using molecular simulation. The tutorial will cover:

- loading in the data set to estimate, and the force field parameters to use in the calculations.
- defining custom calculation schemas for the properties in our data set.
- estimating the data set of properties using an *Evaluator server* instance.
- retrieving the results from the server and storing them on disk.

Note: If you are running this tutorial in google colab you will need to run a setup script instead of following the installation instructions:

[1]:
```
# !wget https://raw.githubusercontent.com/openforcefield/openff-evaluator/master/docs/
↪tutorials/colab_setup.ipynb
# %run colab_setup.ipynb
```

For this tutorial make sure that you are using a GPU accelerated runtime.

For the sake of clarity all warnings will be disabled in this tutorial:

[2]:
```
import warnings
warnings.filterwarnings('ignore')
import logging
logging.getLogger("openforcefield").setLevel(logging.ERROR)
```

We will also enable time-stamped logging to help track the progress of our calculations:

[3]:
```
from openff.evaluator.utils import setup_timestamp_logging
setup_timestamp_logging()
```

2.6.1 Loading the Data Set and Force Field Parameters

We will begin by loading in the data set which we created in the previous tutorial:

[4]:
```
from openff.evaluator.datasets import PhysicalPropertyDataSet

data_set_path = "filtered_data_set.json"

# If you have not yet completed that tutorial or do not have the data set file
# available, a copy is provided by the framework:

# from openff.evaluator.utils import get_data_filename
# data_set_path = get_data_filename("tutorials/tutorial01/filtered_data_set.json")

data_set = PhysicalPropertyDataSet.from_json(data_set_path)
```

As a reminder, this data contains the experimentally measured density and H_{vap} measurements for ethanol and isopropanol at ambient conditions:

[5]:
```
data_set.to_pandas().head()
```

[5]:
```
   Temperature (K)  ...                          Source
0           298.15  ...      10.1016/j.fluid.2013.10.034
1           298.15  ...                10.1021/je1013476
2           298.15  ...  10.1016/S0021-9614(71)80108-8
3           298.15  ...  10.1016/S0021-9614(71)80108-8

[4 rows x 13 columns]
```

We will also define the set of force field parameters which we wish to use to estimate this data set of properties. The framework has support for estimating force field parameters from a range of sources, including those in the OpenFF SMIRNOFF format, those which can be applied by AmberTools, *and more*.

Each source of a force field has a corresponding source object in the framework. In this tutorial we will be using the OpenFF Parsley force field which is based off of the SMIRNOFF format:

[6]:
```
from openff.evaluator.forcefield import SmirnoffForceFieldSource

force_field_path = "openff-1.0.0.offxml"
force_field_source = SmirnoffForceFieldSource.from_path(force_field_path)
```

2.6.2 Defining the Calculation Schemas

The next step we will take will be to define a custom calculation schema for each type of property in our data set.

A calculation schema is the blueprint for how a type of property should be calculated using a particular *calculation approach*, such as directly by simulation, by reprocessing cached simulation data or, in future, a range of other options.

The framework has built-in schemas defining how densities and H_{vap} should be estimated from molecular simulation, covering all aspects from coordinate generation, force field assignment, energy minimisation, equilibration and finally the production simulation and data analysis. All of this functionality is implemented via the frameworks built-in, lightweight *workflow engine*, however we won't dive into the details of this until a later tutorial.

For the purpose of this tutorial, we will simply modify the default calculation schemas to reduce the number of molecules to include in our simulations to speed up the calculations. This step can be skipped entirely if the default options (which we recommend using for 'real-world' calculations) are to be used:

[7]:
```
from openff.evaluator.properties import Density, EnthalpyOfVaporization

density_schema = Density.default_simulation_schema(n_molecules=256)
h_vap_schema = EnthalpyOfVaporization.default_simulation_schema(n_molecules=256)
```

We could further use this method to set either the absolute or the relative uncertainty that the property should be estimated to within. If either of these are set, the simulations will automatically be extended until the target uncertainty in the property has been met.

For our purposes however we won't set any targets, leaving the simulations to run for the default 1 ns.

To use these custom schemas, we need to add them to the a request options object which defines all of the options for estimating our data set:

[8]:
```
from openff.evaluator.client import RequestOptions

# Create an options object which defines how the data set should be estimated.
```

(continues on next page)

(continued from previous page)

```
estimation_options = RequestOptions()
# Specify that we only wish to use molecular simulation to estimate the data set.
estimation_options.calculation_layers = ["SimulationLayer"]

# Add our custom schemas, specifying that the should be used by the 'SimulationLayer'
estimation_options.add_schema("SimulationLayer", "Density", density_schema)
estimation_options.add_schema("SimulationLayer", "EnthalpyOfVaporization", h_vap_schema)
```

2.6.3 Launching the Server

The framework is split into two main applications - an EvaluatorServer and an EvaluatorClient.

The EvaluatorServer is the main object which will perform any and all calculations needed to estimate sets of properties. It is design to run on whichever compute resources you may have available (whether that be a single machine or a high performance cluster), wait until a user requests a set of properties be estimated, and then handle that request.

The EvaluatorClient is the object used by the user to send requests to estimate data sets to running server instances over a TCP connection. It is also used to query the server to see when that request has been fulfilled, and to pull back any results.

Let us begin by spawning a new server instance.

To launch a server, we need to define how this object is going to interact with the compute resource it is running on.

This is accomplished using a *calculation backend*. While there are several to choose from depending on your needs, well will go with a simple dask based one designed to run on a single machine:

[9]:
```
from openff.evaluator.backends import ComputeResources
from openff.evaluator.backends.dask import DaskLocalCluster

calculation_backend = DaskLocalCluster(
    number_of_workers=1,
    resources_per_worker=ComputeResources(
        number_of_threads=1,
        number_of_gpus=1,
        preferred_gpu_toolkit=ComputeResources.GPUToolkit.CUDA
    ),
)
calculation_backend.start()
```

Here we have specified that we want to run our calculations on a single worker which has access to a single GPU.

With that defined, we can go ahead and spin up the server:

[10]:
```
from openff.evaluator.server import EvaluatorServer

evaluator_server = EvaluatorServer(calculation_backend=calculation_backend)
evaluator_server.start(asynchronous=True)
```

```
02:47:53.961 INFO     Server listening at port 8000
```

The server will run asynchronously in the background waiting until a client connects and requests that a data set be estimated.

2.6.4 Estimating the Data Set

With the server spun up we can go ahead and connect to it using an `EvaluatorClient` and request that it estimate our data set using the custom options we defined earlier:

[11]:
```python
from openff.evaluator.client import EvaluatorClient
evaluator_client = EvaluatorClient()

request, exception = evaluator_client.request_estimate(
    property_set=data_set,
    force_field_source=force_field_source,
    options=estimation_options,
)

assert exception is None
```

```
02:47:54.012 INFO     Received estimation request from ('127.0.0.1', 50618)
```

The server will now receive the requests and begin whirring away fulfilling it. It should be noted that the `request_estimate()` function returns two values - a `request` object, and an `exception` object. If all went well (as it should do here) the `exception` object will be `None`.

The `request` object represents the request which we just sent to the server. It stores the unique id which the server assigned to the request, as well as the address of the server that the request was sent to.

The `request` object is primarily used to query the current state of our request, and to pull down the results when it the request finishes. Here we will use it it synchronously query the server every 30 seconds until our request has completed.

[12]:
```python
# Wait for the results.
results, exception = request.results(synchronous=True, polling_interval=30)
assert exception is None
```

Note: we could also asynchronously query for the results of the request. The resultant results object would then contain the partial results of any completed estimates, as well as any exceptions raised during the estimation.

2.6.5 Inspecting the Results

Now that the server has finished estimating our data set and returned the results to us, we can begin to inspect the results of the calculations:

[13]:
```python
print(len(results.queued_properties))

print(len(results.estimated_properties))

print(len(results.unsuccessful_properties))
print(len(results.exceptions))
```

```
0
4
0
0
```

We can (hopefully) see here that there were no exceptions raised during the calculation, and that all of our properties were successfully estimated.

We will extract the estimated data set and save this to disk:

[14]:
```
results.estimated_properties.json("estimated_data_set.json", format=True);
```

2.6.6 Conclusion

And that concludes the second tutorial. In the next tutorial we will be performing some basic analysis on our estimated results.

If you have any questions and / or feedback, please open an issue on the GitHub issue tracker.

2.7 Tutorial 03 - Analysing Data Sets

In this tutorial we will be analysing the results of the calculations which we performed in the *second tutorial*. The tutorial will cover:

- comparing the estimated data set with the experimental data set.
- plotting the two data sets.

Note: If you are running this tutorial in google colab you will need to run a setup script instead of following the installation instructions:

[1]:
```
# !wget https://raw.githubusercontent.com/openforcefield/openff-evaluator/master/docs/
↪tutorials/colab_setup.ipynb
# %run colab_setup.ipynb
```

For the sake of clarity all warnings will be disabled in this tutorial:

[2]:
```
import warnings
warnings.filterwarnings('ignore')
import logging
logging.getLogger("openforcefield").setLevel(logging.ERROR)
```

2.7.1 Loading the Data Sets

We will begin by loading both the experimental data set and the estimated data set:

[3]:
```
from openff.evaluator.datasets import PhysicalPropertyDataSet

experimental_data_set_path = "filtered_data_set.json"
estimated_data_set_path = "estimated_data_set.json"

# If you have not yet completed the previous tutorials or do not have the data set files
# available, copies are provided by the framework:

# from openff.evaluator.utils import get_data_filename
# experimental_data_set_path = get_data_filename(
#     "tutorials/tutorial01/filtered_data_set.json"
# )
# estimated_data_set_path = get_data_filename(
```

(continues on next page)

(continued from previous page)

```
#     "tutorials/tutorial02/estimated_data_set.json"
# )

experimental_data_set = PhysicalPropertyDataSet.from_json(experimental_data_set_path)
estimated_data_set = PhysicalPropertyDataSet.from_json(estimated_data_set_path)
```

if everything went well from the previous tutorials, these data sets will contain the density and H_{vap} of ethanol and isopropanol:

[4]:
```
experimental_data_set.to_pandas().head()
```

[4]:
```
   Temperature (K)  ...                            Source
0           298.15  ...     10.1016/j.fluid.2013.10.034
1           298.15  ...               10.1021/je1013476
2           298.15  ...   10.1016/S0021-9614(71)80108-8
3           298.15  ...   10.1016/S0021-9614(71)80108-8

[4 rows x 13 columns]
```

[5]:
```
estimated_data_set.to_pandas().head()
```

[5]:
```
   Temperature (K)  ...           Source
0           298.15  ...  SimulationLayer
1           298.15  ...  SimulationLayer
2           298.15  ...  SimulationLayer
3           298.15  ...  SimulationLayer

[4 rows x 13 columns]
```

2.7.2 Extracting the Results

We will now compare how the value of each property estimated by simulation deviates from the experimental measurement.

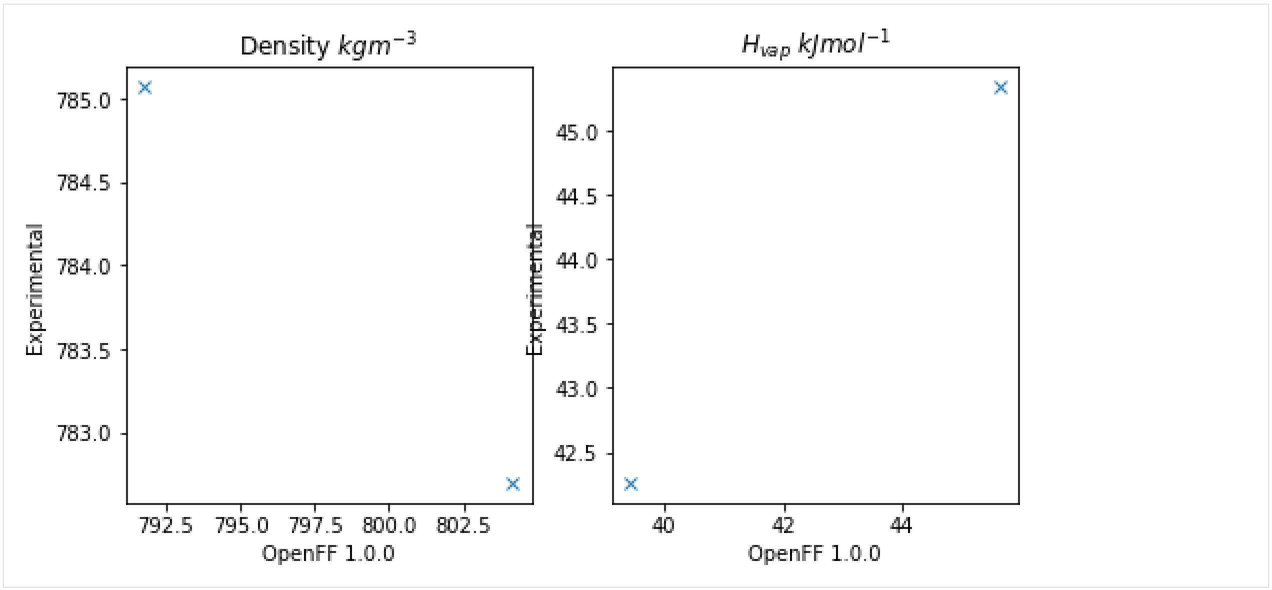
To do this we will extract a list which contains pairs of experimental and evaluated properties. We can easily match properties based on the unique ids which were automatically assigned to them on their creation:

[6]:
```
properties_by_type = {
    "Density": [],
    "EnthalpyOfVaporization": []
}

for experimental_property in experimental_data_set:

    # Find the estimated property which has the same id as the
    # experimental property.
    estimated_property = next(
        x for x in estimated_data_set if x.id == experimental_property.id
    )

    # Add this pair of properties to the list of pairs
    property_type = experimental_property.__class__.__name__
    properties_by_type[property_type].append((experimental_property, estimated_property))
```

2.7.3 Plotting the Results

We will now compare the experimental results to the estimated ones by plotting them using `matplotlib`:

[7]:
```python
from matplotlib import pyplot

# Create the figure we will plot to.
figure, axes = pyplot.subplots(nrows=1, ncols=2, figsize=(8.0, 4.0))

# Set the axis titles
axes[0].set_xlabel('OpenFF 1.0.0')
axes[0].set_ylabel('Experimental')
axes[0].set_title('Density $kg m^{-3}$')

axes[1].set_xlabel('OpenFF 1.0.0')
axes[1].set_ylabel('Experimental')
axes[1].set_title('$H_{vap}$ $kJ mol^{-1}$')

# Define the preferred units of the properties
from openff.evaluator import unit

preferred_units = {
    "Density": unit.kilogram / unit.meter ** 3,
    "EnthalpyOfVaporization": unit.kilojoule / unit.mole
}

for index, property_type in enumerate(properties_by_type):

    experimental_values = []
    estimated_values = []

    preferred_unit = preferred_units[property_type]

    # Convert the values of our properties to the preferred units.
    for experimental_property, estimated_property in properties_by_type[property_type]:

        experimental_values.append(
            experimental_property.value.to(preferred_unit).magnitude
        )
        estimated_values.append(
            estimated_property.value.to(preferred_unit).magnitude
        )

    axes[index].plot(
        estimated_values, experimental_values, marker='x', linestyle='None'
    )
```



2.7.4 Conclusion

And that concludes the third tutorial!

If you have any questions and / or feedback, please open an issue on the GitHub issue tracker.

2.8 Tutorial 04 - Optimizing Force Fields

In this tutorial we will be using the OpenFF Evaluator framework in combination with the fantastic ForceBalance software to optimize a molecular force field against the physical property data set we created in the *first tutorial*.

ForceBalance offers a suite of tools for optimizing molecular force fields against a set of target data. Perhaps one of the most fundamental targets to fit against is experimental physical property data. Physical property data has been used extensively for decades to inform the values of non-bonded Van der Waals (VdW) interaction parameters (often referred to as Lennard-Jones parameters).

ForceBalance is seamlessly integrated with the evaluator framework, using it to evaluate the deviations between target experimentally measured data points and those evaluated using the force field being optimized (as well as the gradient of those deviations with respect to the force field parameters being optimized).

The tutorial will cover:

- setting up the input files and directory structure required by ForceBalace.
- setting up an `EvaluatorServer` for *ForceBalance* to connect to.
- running *ForceBalance* using those input files.
- extracting and plotting a number of statistics output during the optimization.

Note: If you are running this tutorial in google colab you will need to run a setup script instead of following the installation instructions:

[1]:
```
# !wget https://raw.githubusercontent.com/openforcefield/openff-evaluator/master/docs/
↪tutorials/colab_setup.ipynb
# %run colab_setup.ipynb
```

For this tutorial make sure that you are using a GPU accelerated runtime.

For the sake of clarity all warnings will be disabled in this tutorial:

[2]:
```
import warnings
warnings.filterwarnings('ignore')
import logging
logging.getLogger("openforcefield").setLevel(logging.ERROR)
```

We will also enable time-stamped logging to help track the progress of our calculations:

[3]:
```
from openff.evaluator.utils import setup_timestamp_logging
setup_timestamp_logging()
```

2.8.1 Setting up the ForceBalance Inputs

In this section we will be creating the directory structure required by *ForceBalance*, and populating it with the required input files.

Creating the Directory Structure

To begin with, we will create a directory to store the starting force field parameters in:

[4]:
```
!mkdir forcefield
```

and one to store the input parameters for our 'fitting target' - in this case a data set of physical properties:

[5]:
```
!mkdir -p targets/pure_data
```

Defining the Training Data Set

With the directories created, we will next specify the data set of physical properties which we will be training the force field against:

[6]:
```
# For convenience we will use the copy shipped with the framework
from openff.evaluator.utils import get_data_filename
data_set_path = get_data_filename("tutorials/tutorial01/filtered_data_set.json")

# Load the data set.
from openff.evaluator.datasets import PhysicalPropertyDataSet
data_set = PhysicalPropertyDataSet.from_json(data_set_path)

# Due to a small bug in ForceBalance we need to zero out any uncertainties
# which are undefined. This will be fixed in future versions.
from openff.evaluator.attributes import UNDEFINED

for physical_property in data_set:
```

(continues on next page)

(continued from previous page)

```
    if physical_property.uncertainty != UNDEFINED:
        continue

    physical_property.uncertainty = 0.0 * physical_property.default_unit()
```

To speed up the runtime of this tutorial, we will only train the force field against measurements made for ethanol

```
[7]: data_set.filter_by_smiles("CCO")
```

in real optimizations however the data set should be **much** larger than two data points!

With those changes made, we can save the data set in our targets directory:

```
[8]: # Store the data set in the `pure_data` targets folder:
data_set.json("targets/pure_data/training_set.json");
```

Defining the Starting Force Field Parameters

We will use the OpenFF Parsley 1.0.0 force field as the starting parameters for the optimization. These can be loaded directly into an OpenFF `ForceField` object using the OpenFF toolkit:

```
[9]: from openforcefield.typing.engines.smirnoff import ForceField
force_field = ForceField('openff-1.0.0.offxml')
```

In order to use these parameters in *ForceBalance*, we need to 'tag' the individual parameters in the force field that we wish to optimize. The toolkit easily enables us to add these tags using cosmetic attributes:

```
[10]: # Extract the smiles of all unique components in our data set.
from openforcefield.topology import Molecule, Topology

all_smiles = set(
    component.smiles
    for substance in data_set.substances
    for component in substance.components
)

for smiles in all_smiles:

    # Find those VdW parameters which would be applied to those components.
    molecule = Molecule.from_smiles(smiles)
    topology = Topology.from_molecules([molecule])

    labels = force_field.label_molecules(topology)[0]

    # Tag the exercised parameters as to be optimized.
    for parameter in labels["vdW"].values():
        parameter.add_cosmetic_attribute("parameterize", "epsilon, rmin_half")
```

Here we have made use of the toolkit's handy `label_molecules` function to see which VdW parameters will be assigned to the molecules in our data set, and tagged them to be parameterized.

With those tags added, we can save the parameters in the `forcefield` directory:

[11]:
```
# Save the annotated force field file.
force_field.to_file('forcefield/openff-1.0.0-tagged.offxml')
```

*Note: The force field parameters are stored in the*OpenFF SMIRNOFF XML format.

Creating the Main Input File

Next, we will create the main *ForceBalance* input file. For the sake of brevity a default input file which ships with this framework will be used:

[12]:
```
input_file_path = get_data_filename("tutorials/tutorial04/optimize.in")

# Copy the input file into our directory structure
import shutil
shutil.copyfile(input_file_path, "optimize.in")
```

[12]:
```
'optimize.in'
```

While there are many options that can be set within this file, the main options of interest for our purposes appear at the bottom of the file:

[13]:
```
!tail -n 6 optimize.in
```

```
$target
name pure_data
type Evaluator_SMIRNOFF
weight 1.0
openff.evaluator_input options.json
$end
```

Here we have specified that we wish to create a new *ForceBalance* `Evaluator_SMIRNOFF` target called `pure_data` (corresponding to the name of the directory we created in the earlier step).

The main input to this target is the file path to an `options.json` file - it is this file which will specify all the options which should be used when *ForceBalance* requests that our target data set be estimated using the current sets of force field parameters.

We will create this file in the `targets/pure_data` directory later in this section.

The data set is the JSON serialized representation of the `PhysicalPropertyDataSet` we created during the *first tutorial*.

Defining the Estimation Options

The final step before we can start the optimization is to create the set of options which will govern how our data set is estimated using the Evaluator framework.

These options will be stored in an `Evaluator_SMIRNOFF` object:

[14]:
```
from forcebalance.evaluator_io import Evaluator_SMIRNOFF

# Create the ForceBalance options object
target_options = Evaluator_SMIRNOFF.OptionsFile()
# Set the path to the data set
target_options.data_set_path = "training_set.json"
```

This object exposes both a set of *ForceBalance* specific options, as well as the set of Evaluator options.

The *ForceBalance* specific options allow us to define how each type of property will contribute to the optimization objective function (the value which we are trying to minimize):

$$\Delta(\theta) = \sum_{n}^{N} \frac{weight_n}{M_n} \sum_{m}^{M_n} \left(\frac{y_m^{ref} - y_m(\theta)}{denominator_n} \right)^2$$

where N is the number of types of properties (e.g. density, enthalpy of vaporization, etc.), M_n is the number of data points of type n, y_m^{ref} is the experimental value of data point m and $y_m(\theta)$ is the estimated value of data point m using the current force field parameters

In particular, the options object allows us to specify both an amount to scale each type of properties contribution to the objective function by ($weight_n$), and the amount to scale the difference between the experimental and estimated properties ($denominator_n$):

[15]:
```python
from openff.evaluator import unit

target_options.weights = {
    "Density": 1.0,
    "EnthalpyOfVaporization": 1.0
}
target_options.denominators = {
    "Density": 30.0 * unit.kilogram / unit.meter ** 3,
    "EnthalpyOfVaporization": 3.0 * unit.kilojoule / unit.mole
}
```

where here we have chosen values that ensure that both types of properties contribute roughly equally to the total objective function.

The Evaluator specific options correspond to a standard `RequestOptions` object:

[16]:
```python
from openff.evaluator.client import RequestOptions

# Create the options which evaluator should use.
evaluator_options = RequestOptions()
# Choose which calculation layers to make available.
evaluator_options.calculation_layers = ["SimulationLayer"]

# Reduce the default number of molecules
from evaluator.properties import Density, EnthalpyOfVaporization

density_schema = Density.default_simulation_schema(n_molecules=256)
h_vap_schema = EnthalpyOfVaporization.default_simulation_schema(n_molecules=256)

evaluator_options.add_schema("SimulationLayer", "Density", density_schema)
evaluator_options.add_schema("SimulationLayer", "EnthalpyOfVaporization", h_vap_schema)

target_options.estimation_options = evaluator_options
```

These options allow us to control exactly how each type of property should be estimated, which calculation approaches should be used and more. Here we use the same options are were used in the *second tutorial*

Note: more information about the different estimation options can befound here

And that's the options created! We will finish off by serializing the options into our target directory:

[17]:
```
# Save the options to file.
with open("targets/pure_data/options.json", "w") as file:
    file.write(target_options.to_json())
```

2.8.2 Launching an Evaluator Server

With the *ForceBalance* options created, we can now move onto launching the `EvaluatorServer` which *ForceBalance* will call out to when it needs the data set to be evaluated:

[18]:
```
# Launch the calculation backend which will distribute any calculations.
from openff.evaluator.backends import ComputeResources
from openff.evaluator.backends.dask import DaskLocalCluster

calculation_backend = DaskLocalCluster(
    number_of_workers=1,
    resources_per_worker=ComputeResources(
        number_of_threads=1,
        number_of_gpus=1,
        preferred_gpu_toolkit=ComputeResources.GPUToolkit.CUDA
    ),
)
calculation_backend.start()

# Launch the server object which will listen for estimation requests and schedule any
# required calculations.
from openff.evaluator.server import EvaluatorServer

evaluator_server = EvaluatorServer(calculation_backend=calculation_backend)
evaluator_server.start(asynchronous=True)
```
```
01:30:20.505 INFO     Server listening at port 8000
```

We will not go into the details of this here as this was already covered in the *second tutorial*

2.8.3 Running ForceBalance

With the inputs created and an Evaluator server spun up, we are finally ready to run the optimization! This can be accomplished with a single command:

[19]:
```
!ForceBalance optimize.in
```

If everything went well *ForceBalance* should exit cleanly, and will have stored out newly optimized force field in the `results` directory.

[20]:
```
!ls result/optimize
```
```
openff-1.0.0-tagged_1.offxml  openff-1.0.0-tagged.offxml
```

2.8.4 Plotting the results

As a last step in this tutorial, we will extract the objective function at each iteration from the *ForceBalance* output files and plot this using `matplotlib`.

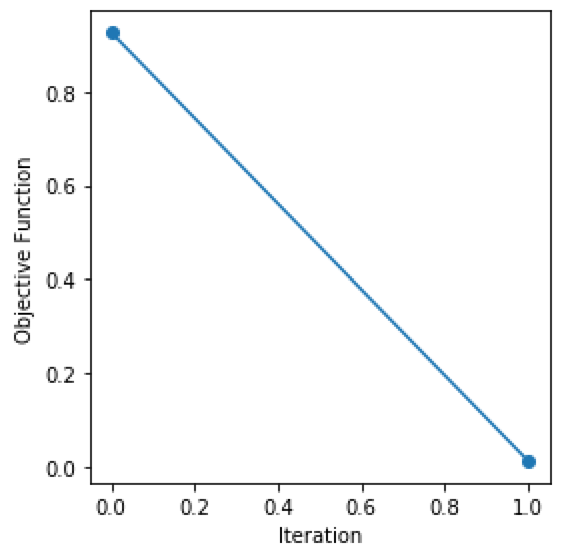
First, we will extract the objective function from the `pickle` serialized output files which can be found in the `optimize.tmp/pure_data/iter_****/` directories:

[21]:
```
from forcebalance.nifty import lp_load

# Determine how many iterations ForceBalance has completed.
from glob import glob
n_iterations = len(glob("optimize.tmp/pure_data/iter*"))

# Extract the objective function at each iteration.
objective_function = []

for iteration in range(n_iterations):

    folder_name = "iter_" + str(iteration).zfill(4)
    file_path = f"optimize.tmp/pure_data/{folder_name}/objective.p"

    statistics = lp_load(file_path)
    objective_function.append(statistics["X"])

print(objective_function)
```

```
[0.9270359101845124, 0.011497456194198362]
```

The objective function is then easily plotted:

[22]:
```
from matplotlib import pyplot
figure, axis = pyplot.subplots(1, 1, figsize=(4, 4))

axis.set_xlabel("Iteration")
axis.set_ylabel("Objective Function")

axis.plot(range(n_iterations), objective_function, marker="o")

figure.tight_layout()
```



2.8.5 Conclusion

And that concludes the fourth tutorial!

If you have any questions and / or feedback, please open an issue on the GitHub issue tracker.

2.9 Property Data Sets

A *PhysicalPropertyDataSet* is a collection of measured physical properties encapsulated as *physical property* objects. They may be created from scratch:

```
# Define a density measurement
density = Density(
    substance=Substance.from_components("O"),
    thermodynamic_state=ThermodynamicState(
        pressure=1.0*unit.atmospheres, temperature=298.15*unit.kelvin
    ),
    phase=PropertyPhase.Liquid,
    value=1.0*unit.gram/unit.millilitre,
    uncertainty=0.0001*unit.gram/unit.millilitre
)

# Add the property to a data set
data_set = PhysicalPropertyDataset()
data_set.add_properties(density)
```

are readily JSON (de)serializable:

```
# Save the data set as a JSON file.
data_set.json(file_path="data_set.json", format=True)
# Load the data set from a JSON file
data_set = PhysicalPropertyDataset.from_json(file_path="data_set.json")
```

and may be converted to pandas `DataFrame` objects:

```
data_set.to_pandas()
```

The framework implements specific data set objects for extracting data measurements directly from a number of open data sources, such as the *ThermoMLDataSet* (see *ThermoML Archive*) which provides utilities for extracting the data from the NIST ThermoML Archive and converting it into the standard framework objects.

Data set objects are directly iterable:

```
for physical_property in data_set:
    ...
```

or can be iterated over for a specific substance:

```
for physical_property in data_set.properties_by_substance(substance):
    ...
```

or for a specific type of property:

```
for physical_property in data_set.properties_by_type("Density"):
    ...
```

2.9.1 Physical Properties

The *PhysicalProperty* object is a base class for any object which describes a measured property of substance, and is defined by a combination of:

- the observed value of the property.
- *Substance* specifying the substance that the measurement was collected for.
- *PropertyPhase* specifying the phase that the measurement was collected in.
- *ThermodynamicState* specifying the thermodynamic conditions under which the measurement was performed

as well as optionally

- the uncertainty in the value of the property.
- a list of *ParameterGradient* which defines the gradient of the property with respect to the model parameters if it was computationally estimated.
- a *Source* specifying the source (either experimental or computational) and provenance of the measurement.

Each type of property supported by the framework, such as a density of an enthalpy of vaporization, must have it's own class representation which inherits from *PhysicalProperty*:

```
# Define a density measurement
density = Density(
    substance=Substance.from_components("O"),
    thermodynamic_state=ThermodynamicState(
        pressure=1.0*unit.atmospheres, temperature=298.15*unit.kelvin
    ),
    phase=PropertyPhase.Liquid,
    value=1.0*unit.gram/unit.millilitre,
    uncertainty=0.0001*unit.gram/unit.millilitre
)
```

2.9.2 Substances

A *Substance* is defined by a number of components (which may have specific roles assigned to them such as being solutes in the system) and the amount of each component in the substance.

To create a pure substance containing only water:

```
water_substance = Substance.from_components("O")
```

To create binary mixture of water and methanol in a 20:80 ratio:

```
binary_mixture = Substance()
binary_mixture.add_component(Component(smiles="O"), MoleFraction(value=0.2))
binary_mixture.add_component(Component(smiles="CO"), MoleFraction(value=0.8))
```

To create a substance of an infinitely dilute paracetamol solute dissolved in water:

```
solution = Substance()
solution.add_component(
    Component(smiles="O", role=Component.Role.Solvent), MoleFraction(value=1.0)
)
solution.add_component(
    Component(smiles="CC(=O)Nc1ccc(O)cc1", role=Component.Role.Solute),␣
↪ExactAmount(value=1)
)
```

2.9.3 Property Phases

The *PropertyPhase* enum describes the possible phases which a measurement was performed in.

While the enum only has three defined phases (`Solid`, `Liquid` and `Gas`), multiple phases can be formed by OR'ing (|) multiple phases together. As an example, to define a phase for a liquid and gas coexisting:

```
liquid_gas_phase = PropertyPhase.Liquid | PropertyPhase.Gas
```

2.9.4 Thermodynamic States

A *ThermodynamicState* specifies a combination of the temperature and (optionally) the pressure at which a measurement is performed:

```
thermodynamic_state = ThermodynamicState(
    temperature=298.15*unit.kelvin, pressure=1.0*unit.atmosphere
)
```

2.10 ThermoML Archive

The *ThermoMLDataSet* object offers an API for extracting physical properties from the NIST ThermoML Archive, both directly from the archive itself or from files stored in the IUPAC- standard ThermoML format.

The API only supports extracting those properties which have been *registered* with the frameworks plug-in system, and does not currently load the full set of metadata available in the archive files.

Note: If the metadata you require is not currently exposed, please open an issue on the GitHub issue tracker to request it.

Currently the framework has built-in support for extracting:

- *Mass density, kg/m3* (Density)
- *Excess molar volume, m3/mol* (ExcessMolarVolume)
- *Relative permittivity at zero frequency* (DielectricConstant)
- *Excess molar enthalpy (molar enthalpy of mixing), kJ/mol* (EnthalpyOfMixing)
- *Molar enthalpy of vaporization or sublimation, kJ/mol* (EnthalpyOfVaporization)

where here both the ThermoML property name (as defined by the IUPAC XML schema) and the built-in framework class are listed.

2.10.1 Registering Properties

Properties to be extracted from ThermoML archives must have a corresponding class representation to be loading into. This class representation must both:

- inherit from the frameworks *PhysicalProperty* class and
- be registered with the frameworks plug-in system using either the *thermoml_property()* decorator or the *register_thermoml_property()* method.

As an example, a class representation of the ThermoML *'Mass density, kg/m3'* property could be defined and registered with the plug-in system using:

```
@thermoml_property("Mass density, kg/m3", supported_phases=PropertyPhase.Liquid)
class Density(PhysicalProperty):
    """A class representation of a mass density property"""
```

The *thermoml_property()* decorator takes in the name of the ThermoML property (as defined by the IUPAC schema) as well as the phases which the framework will be able to estimate this property in.

Multiple ThermoML properties can be mapped onto a single class using the flexible *register_thermoml_property()* function. For example, the *'Specific volume, m3/kg'* property (which is simply the reciprocal of mass density) may be mapped onto the *Density* object by providing a `conversion_function`:

```
def specific_volume_to_mass_density(specific_volume):
    """Converts a specific volume measurement into a mass
    density.

    Parameters
    ----------
```

(continues on next page)

(continued from previous page)

```
    specific_volume: ThermoMLProperty
        The specific volume measurement to convert.
    """
    mass_density = Density()

    mass_density.value = 1.0 / specific_volume.value

    if mass_density.uncertainty is not None:
        mass_density.uncertainty = 1.0 / mass_density.uncertainty

    mass_density.phase = specific_volume.phase

    mass_density.thermodynamic_state = specific_volume.thermodynamic_state
    mass_density.substance = specific_volume.substance

    return mass_density

# Register the ThermoML property using the conversion function.
register_thermoml_property(
    thermoml_string="Specific volume, m3/kg",
    supported_phases=PropertyPhase.Liquid,
    property_class=Density,
    conversion_function=specific_volume_to_mass_density
)
```

Converting the different density derivatives into a single density class removes the need to produce many very similar class representations of density measurements, and allows a single calculation schema to be defined for all variants.

2.10.2 Loading Data Sets

Data sets are most easily loaded using their digital object identifiers (DOI). For example, to retrieve the ThermoML data set that accompanies this paper, we can simply use the DOI `10.1016/j.jct.2005.03.012`:

```
data_set = ThermoMLDataset.from_doi('10.1016/j.jct.2005.03.012')
```

Data can be pulled from multiple sources at once by specifying multiple identifiers:

```
identifiers = ['10.1021/acs.jced.5b00365', '10.1021/acs.jced.5b00474']
dataset = ThermoMLDataset.from_doi(*identifiers)
```

Entire archives of properties can be downloaded directly from the ThermoML website and parsed by the framework. For example, to create a data set object containing all of the measurements recorded from the International Journal of Thermophysics:

```
# Download the archive of all properties from the IJT journal.
import requests
request = requests.get("https://trc.nist.gov/ThermoML/IJT.tgz", stream=True)

# Make sure the request went ok.
assert request

# Unzip the files into a new 'ijt_files' directory.
```

(continues on next page)

(continued from previous page)

```
import io, tarfile
tar_file = tarfile.open(fileobj=io.BytesIO(request.content))
tar_file.extractall("ijt_files")

# Get the names of the extracted files
import glob
file_names = glob.glob("ijt_files/*.xml")

# Create the data set object
from openff.evaluator.datasets.thermoml import ThermoMLDataSet
data_set = ThermoMLDataSet.from_file(*file_names)

# Save the data set to a JSON object
data_set.json(file_path="ijt.json", format=True)
```

2.11 Taproom

The *TaproomDataSet* object offers an API for retrieving host-guest binding affinity measurements from the curated taproom repository.

Note: `taproom` may be installed by running `conda install -c conda-forge taproom`

This includes retrieving all of the data available:

```
from openff.evaluator.datasets.taproom import TaproomDataSet
taproom_set = TaproomDataSet()
```

data measure for a single host molecule (e.g. alpha-cyclodextrin):

```
acd_taproom_set = TaproomDataSet(host_codes=["acd"])
```

or data for a particular host and guest pair:

```
acd_taproom_set = TaproomDataSet(host_codes=["acd"], guest_codes=["bam"])
```

All measurements in this data set have an associated *TaproomSource* as their source provenance. This tracks both the original source of the measurement as well as the taproom identifier.

Note: Currently the data set object will assume a default set of buffer conditions (either no buffer, or a buffer of a salt with a specified ionic strength) rather than reading the buffer from the taproom measurement directory. This is consistent with previous applications of the data set.

2.12 Data Set Curation

The framework offers a full suite of features to facilitate the curation of data sets of physical properties, including:

- a significant amount of data filters, including to filter by state, substance composition and chemical functionalities.

and components to

- easily download and import the full *NIST ThermoML* and FreeSolv archives .
- select data points which were measured close to a set of target states, and which were measured for a diverse range of substances which contain specific functionalities.
- convert between different compatible property types (e.g. convert density <-> excess molar volume data).

These features are implemented as `CurationComponent` objects, which take as input an associated `CurationComponentSchema` which controls how the curation components should be applied to a particular data set (or a data set which is being stored as pandas `DataFrame` object).

An example of a curation component would be one that filters out data points which were measured outside of a particular temperature range:

```
# Filter data points measured at less than 290.0 K or greater than 320.0 K
filtered_frame = FilterByTemperature.apply(
    data_frame,
    FilterByTemperatureSchema(minimum_temperature=290.0, maximum_temperature=320.0),
)
```

Curation components can be conveniently chained together using a `CurationWorkflow` and an associated `CurationWorkflowSchema` so as to easily curated full training and testing data sets:

```
curation_schema = WorkflowSchema(
    component_schemas=[
        # Import the ThermoML archive.
        thermoml.ImportThermoMLDataSchema()
        # Filter out any measurements made for systems with more than two components
        filtering.FilterByNComponentsSchema(n_components=[1, 2]),
        # Remove any duplicate data.
        filtering.FilterDuplicatesSchema(),
        # Filter out data points measured away from ambient
        # and biologically relevant temperatures.
        filtering.FilterByTemperatureSchema(
            minimum_temperature=298.0, maximum_temperature=320.0
        ),
        # Retain only density and enthalpy of mixing data points.
        filtering.FilterByPropertyTypesSchema(
            property_types=["Density", "EnthalpyOfMixing"],
        ),
        # Select data points measured for alcohols, esters or mixtures of both.
        selection.SelectSubstancesSchema(
            target_environments=[
                ChemicalEnvironment.Alcohol,
                ChemicalEnvironment.CarboxylicAcidEster,
            ],
            n_per_environment=10,
```

(continues on next page)

(continued from previous page)

```
        ),
    ]
)

data_frame = Workflow.apply(pandas.DataFrame(), curation)
```

2.12.1 Examples

Data Extraction

- *ImportFreeSolv*: A component which will download the latest, full FreeSolv data set from the GitHub repository:

```
from openff.evaluator.datasets.curation.components.freesolv import (
    ImportFreeSolv,
    ImportFreeSolvSchema,
)

# Import the full FreeSolv data set.
data_frame = ImportFreeSolv.apply(pandas.DataFrame(), ImportFreeSolvSchema())
```

- *ImportThermoMLData*: A component which will download all *supported data* from the NIST ThermoML Archive:

```
from openff.evaluator.datasets.curation.components.thermoml import (
    ImportThermoMLData,
    ImportThermoMLDataSchema,
)

# Import all data collected from the IJT journal.
data_frame = ImportThermoMLData.apply(pandas.DataFrame(),␣
↪ImportThermoMLDataSchema())
```

Filtration

- *FilterDuplicates*: A component to remove duplicate data points (within a specified precision) from a data set:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterDuplicates,
    FilterDuplicatesSchema,
)

filtered_frame = FilterDuplicates.apply(data_frame, FilterDuplicatesSchema())
```

- *FilterByTemperature*: A component which will filter out data points which were measured outside of a specified temperature range:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByTemperature,
```

(continues on next page)

(continued from previous page)

```
    FilterByTemperatureSchema,
)

filtered_frame = FilterByTemperature.apply(
    data_frame,
    FilterByTemperatureSchema(minimum_temperature=290.0, maximum_temperature=320.0),
)
```

- *FilterByPressure*: A component which will filter out data points which were measured outside of a specified pressure range:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByPressure,
    FilterByPressureSchema,
)

filtered_frame = FilterByPressure.apply(
    data_frame,
    FilterByPressureSchema(minimum_pressure=100.0, maximum_pressure=140.0),
)
```

- *FilterByMoleFraction*: A component which will filter out data points which were measured outside of a specified mole fraction range:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByMoleFraction,
    FilterByMoleFractionSchema,
)

filtered_frame = FilterByMoleFraction.apply(
    data_frame, FilterByMoleFractionSchema(mole_fraction_ranges={2: [[(0.1, 0.3)]]})
)
```

- *FilterByRacemic*: A component which will filter out data points which were measured for racemic mixtures:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByRacemic,
    FilterByRacemicSchema,
)

filtered_frame = FilterByRacemic.apply(data_frame, FilterByRacemicSchema())
```

- *FilterByElements*: A component which will filter out data points which were measured for systems which contain specific elements:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByElements,
    FilterByElementsSchema,
)

filtered_frame = FilterByElements.apply(
    data_frame,
```

(continues on next page)

(continued from previous page)

```
    FilterByElementsSchema(allowed_elements=["C", "O", "H"]),
)
```

- *FilterByPropertyTypes*: A component which will apply a filter which only retains properties of specified types:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByPropertyTypes,
    FilterByPropertyTypesSchema,
)

# Retain only density measurements made for either pure or binary systems.
filtered_frame = FilterByPropertyTypes.apply(
    data_frame,
    FilterByPropertyTypesSchema(
        property_types=["Density"],
        n_components={"Density": [1, 2]},
    ),
)
```

- *FilterByStereochemistry*: A component which filters out data points measured for systems whereby the stereochemistry of a number of components is undefined:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByStereochemistry,
    FilterByStereochemistrySchema,
)

filtered_frame = FilterByStereochemistry.apply(
    data_frame, FilterByStereochemistrySchema()
)
```

- *FilterByCharged*: A component which filters out data points measured for substance where any of the constituent components have a net non-zero charge.:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByCharged,
    FilterByChargedSchema,
)

filtered_frame = FilterByCharged.apply(data_frame, FilterByChargedSchema())
```

- *FilterByIonicLiquid*: A component which filters out data points measured for substances which contain or are classed as an ionic liquids:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByIonicLiquid,
    FilterByIonicLiquidSchema,
)

filtered_frame = FilterByIonicLiquid.apply(data_frame, FilterByIonicLiquidSchema())
```

- *FilterBySmiles*: A component which filters the data set so that it only contains either a specific set of smiles, or does not contain any of a set of specifically excluded smiles:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterBySmiles,
    FilterBySmilesSchema,
)

filtered_frame = FilterBySmiles.apply(
    data_frame, FilterBySmilesSchema(smiles_to_include=["CCCO"]),
)
```

- *FilterBySmirks*: A component which filters a data set so that it only contains measurements made for molecules which contain (or don't) a set of chemical environments represented by SMIRKS patterns:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterBySmirks,
    FilterBySmirksSchema,
)

filtered_frame = FilterBySmirks.apply(
    data_frame, FilterBySmirksSchema(smirks_to_include=["[#6a]"]),
)
```

- *FilterByNComponents*: A component which filters out data points measured for systems with specified number of components:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByNComponents,
    FilterByNComponentsSchema,
)

filtered_frame = FilterByNComponents.apply(
    data_frame, FilterByNComponentsSchema(n_components=[1, 2])
)
```

- *FilterBySubstances*: A component which filters the data set so that it only contains properties measured for particular substances:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterBySubstances,
    FilterBySubstancesSchema,
)

filtered_frame = FilterBySubstances.apply(
    data_frame, FilterBySubstancesSchema(substances_to_include=[("CO", "C")])
)
```

- *FilterByEnvironments*: A component which filters a data set so that it only contains measurements made for substances which contain specific chemical environments:

```
from openff.evaluator.datasets.curation.components.filtering import (
    FilterByEnvironments,
    FilterByEnvironmentsSchema,
)
```

(continues on next page)

(continued from previous page)

```
filtered_frame = FilterByEnvironments.apply(
    data_frame,
    FilterByEnvironmentsSchema(
        environments=[
            ChemicalEnvironment.Aqueous,
            ChemicalEnvironment.Alcohol,
            ChemicalEnvironment.Amine,
        ]
    ),
)
```

Data Selection

- *SelectSubstances*: A component for selecting data points which were measured for specified number of maximally diverse systems containing a specified set of chemical functionalities:

```
# Select (if possible) data points which were measured for 10 different (and
# structurally diverse) alcohols.
schema = SelectSubstancesSchema(
    target_environments=[ChemicalEnvironment.Alcohol],
    n_per_environment=10,
)

data_frame = ConvertExcessDensityData.apply(data_frame, schema)
```

- *SelectDataPoints*: A component for selecting a set of data points which are close to a particular set of states:

```
# Select (if possible) density data points which were measured for pure systems
# at close to 298.15 K and 308.15K
schema = SelectDataPointsSchema(
    target_states=[
        TargetState(
            property_types=[("Density", 1)],
            states=[
                State(temperature=298.15, pressure=101.325, mole_fractions=(1.0,)),
                State(temperature=308.15, pressure=101.325, mole_fractions=(1.0,)),
            ],
        )
    ]
)

data_frame = ConvertExcessDensityData.apply(data_frame, schema)
```

Data Conversion

- *ConvertExcessDensityData*: A component for converting binary mass density data to excess molar volume data and vice versa where pure density data measured for the components is available:

```
from openff.evaluator.datasets.curation.components.conversion import (
    ConvertExcessDensityData,
    ConvertExcessDensityDataSchema,
)

converted_data_frame = ConvertExcessDensityData.apply(
    data_frame, ConvertExcessDensityDataSchema()
)
```

2.13 Physical Properties

A core philosophy of this framework is that users should be able to seamlessly curate data sets of physical properties and then estimate that data set using computational methods without significant user intervention and using sensible, well validated workflows.

This page aims to provide an overview of which physical properties are supported by the framework and how they are computed using the different *calculation layers*.

In this document $\langle X \rangle$ will be used to denote the ensemble average of an observable X.

2.13.1 Density

The density (ρ) is computed according to

$$\rho = \left\langle \frac{M}{V} \right\rangle$$

where M and V are the total molar mass and volume the system respectively.

Direct Simulation

The density is estimated using the default *simulation workflow* without modification. The estimation of liquid densities is assumed.

MBAR Reweighting

The density is estimated using the default *reweighting workflow* without modification. The estimation of liquid densities is assumed.

2.13.2 Dielectric Constant

The dielectric constant (ε) is computed from the fluctuations in a systems dipole moment (see Equation 7 of [1]) according to:

$$\varepsilon = 1 + \frac{\left\langle \vec{\mu}^2 \right\rangle - \left\langle \vec{\mu} \right\rangle^2}{3\varepsilon_0 \left\langle V \right\rangle k_b T}$$

where $\vec{\mu}$, V are the systems dipole moment and volume respectively, k_b the Boltzmann constant, T the temperature, and ε_0 the permittivity of free space.

Note: In *v0.2.2* and earlier of the framework the variance was computed as $\left\langle \left(\vec{\mu} - \left\langle \vec{\mu} \right\rangle \right)^2 \right\rangle$ in order to match the mdtraj implementation which has been used in previous studies by the OpenFF Consortium (see for example [2]). The two approaches should be numerically indistinguishable however.

Direct Simulation

The dielectric is estimated using the default *simulation workflow* which has been modified to use the specialized `AverageDielectricConstant` protocol in place of the default `AverageObservable` protocol. The estimation of liquid dielectric constants is assumed.

MBAR Reweighting

The dielectric is estimated using the default *reweighting workflow* which has been modified to use the specialized `ReweightDielectricConstant` protocol in place of the default `ReweightObservable` protocol. It should be noted that the `ReweightDielectricConstant` protocol employs bootstrapping to compute the uncertainty in the average dielectric constant, rather than attempting to propagate uncertainties in the average dipole moments and volumes. The estimation of liquid dielectric constants is assumed.

2.13.3 Enthalpy of Vaporization

The enthalpy of vaporization ΔH_{vap} (see [3]) can be computed according to

$$\Delta H_{vap} = \left\langle H_{gas} \right\rangle - \left\langle H_{liquid} \right\rangle = \left\langle E_{gas} \right\rangle - \left\langle E_{liquid} \right\rangle + p \left(\left\langle V_{gas} \right\rangle - \left\langle V_{liquid} \right\rangle \right)$$

where H, E, and V are the enthalpy, total energy and volume respectively.

Under the assumption that $V_{gas} >> V_{liquid}$ and that the gas is ideal the above expression can be simplified to

$$\Delta H_{vap} = \left\langle U_{gas} \right\rangle - \left\langle U_{liquid} \right\rangle + RT$$

where U is the potential energy, T the temperature and R the universal gas constant. This simplified expression is computed by default by this framework.

Direct Simulation

- **Liquid phase**: The potential energy of the liquid phase is estimated using the default *simulation workflow*, and divided by the number of molecules in the simulation box using the `divisor` input of the `AverageObservable` protocol.
- **Gas phase**: The potential energy of the gas phase is estimated using the default *simulation workflow*, which has been modified so that
  - the simulation box only contains a single molecule.
  - all periodic boundary conditions have been disabled.
  - all simulations are performed in the NVT ensemble.
  - the production simulation is run for 15000000 steps at a time (rather than 1000000 steps).
  - all simulations are run using the OpenMM reference platform (CPU only) regardless of whether a GPU is available. This is fastest platform to use when simulating a single molecule in vacuum with OpenMM.

The final enthalpy is then computed by subtracting the gas potential energy from the liquid potential energy (`SubtractValues`) and adding the RT term (`AddValues`). Uncertainties are propagated through the subtraction by the normal means using the uncertainties package.

MBAR Reweighting

- **Liquid phase**: The potential energy of the liquid phase is estimated using the default *reweighting workflow*, and divided by the number of molecules in the simulation box using an extra `DivideValue` protocol.
- **Gas phase**: The potential energy of the gas phase is estimated using the default *reweighting workflow*, which has been modified so that all periodic boundary conditions have been disabled.

The final enthalpy is then computed by subtracting the gas potential energy from the liquid potential energy (`SubtractValues`) and adding the RT term (`AddValues`). Uncertainties are propagated through the subtraction by the normal means using the uncertainties package.

2.13.4 Enthalpy of Mixing

The enthalpy of mixing $\Delta H_{mix}\left(x_0, \cdots, x_{M-1}\right)$ for a system of M components is computed according to

$$\Delta H_{mix}\left(x_0, \cdots, x_{M-1}\right) = \frac{\langle H_{mix} \rangle}{N_{mix}} - \sum_i^M x_i \frac{\langle H_i \rangle}{N_i}$$

where H_{mix} is the enthalpy of the full mixture, and H_i, x_i are the enthalpy and the mole fraction of component i respectively. N_{mix} and N_i are the total number of molecules used in the full mixture simulations and the simulations of each individual component respectively.

When re-weighting cached data to compute H_{mix} we make the approximation that the kinetic energy contributions cancel out between the mixture and each of the components, and hence can be computed by only re-weighting the NPT reduced potential:

$$\Delta H_{mix}\left(x_0, \cdots, x_{M-1}\right) \approx \frac{1}{\beta} \left(\frac{\langle u_{mix} \rangle}{N_{mix}} - \sum_i^M x_i \frac{\langle u_i \rangle}{N_i} \right)$$

where $u \equiv \beta \left(U + pV\right)$ is the NPT reduced potential, U the potential energy, p the pressure and V the volume.

Direct Simulation

- **Mixture**: The enthalpy of the full mixture is estimated using the default *simulation workflow* and divided by the number of molecules in the simulation box using the `divisor` input of the `AverageObservable` protocol.
- **Components**: The enthalpy of each of the components is estimated using the default *simulation workflow*, divided by the number of molecules in the simulation box using the `divisor` input of the `AverageObservable` protocol, and weighted by their mole fraction *in the mixture simulation box* using the `WeightByMoleFraction` protocol.

The final enthalpy is then computed by summing the component enthalpies (`AddValues`) and subtracting these from the mixture enthalpy (`SubtractValues`). Uncertainties are propagated through the summation and subtraction by the normal means using the uncertainties package.

MBAR Reweighting

- **Mixture**: The reduced potential of the full mixture is estimated using the default *reweighting workflow* and divided by the number of molecules in the reweighting box using an extra `DivideValue` protocol.
- **Components**: The reduced potential of each of the components is estimated using the default *reweighting workflow*, divided by the number of molecules in the reweighting box using an extra `DivideValue` protocol, and weighted by their mole fraction using the `WeightByMoleFraction` protocol.

The final enthalpy is then computed by summing the component enthalpies (`AddValues`), subtracting these from the mixture enthalpy (`SubtractValues`), and multiplying by $1/\beta$ (`MultiplyValue`). Uncertainties are propagated by the normal means using the uncertainties package.

2.13.5 Excess Molar Volume

The excess molar volume $\Delta V_{excess}\left(x_0, \cdots, x_{M-1}\right)$ for a system of M components is computed according to

$$\Delta V_{excess}\left(x_0, \cdots, x_{M-1}\right) = N_A\left(\frac{\langle V_{mix}\rangle}{N_{mix}} - \sum_i^M x_i \frac{\langle V_i\rangle}{N_i}\right)$$

where V_{mix} is the volume of the full mixture, and V_i, x_i are the volume and the mole fraction of component i respectively. N_{mix} and N_i are the total number of molecules used in the full mixture simulations and the simulations of each individual component respectively, and N_A is the Avogadro constant.

Direct Simulation

- **Mixture**: The molar volume of the full mixture is estimated using the default *simulation workflow* and divided by the molar number of molecules in the simulation box using the `divisor` input of the `AverageObservable` protocol.
- **Components**: The molar volume of each of the components is estimated using the default *simulation workflow*, divided by the molar number of molecules in the simulation box using the `divisor` input of the `AverageObservable` protocol, and weighted by their mole fraction *in the mixture simulation box* using the `WeightByMoleFraction` protocol.

The final excess molar volume is then computed by summing the component molar volumes (`AddValues`) and subtracting these from the mixture molar volume (`SubtractValues`). Uncertainties are propagated through the summation and subtraction by the normal means using the uncertainties package.

MBAR Reweighting

- **Mixture**: The enthalpy of the full mixture is estimated using the default *reweighting workflow* and divided by the molar number of molecules in the reweighting box using an extra `DivideValue` protocol.
- **Components**: The enthalpy of each of the components is estimated using the default *reweighting workflow*, divided by the molar number of molecules in the reweighting box using an extra `DivideValue` protocol, and weighted by their mole fraction using the `WeightByMoleFraction` protocol.

The final enthalpy is then computed by summing the component enthalpies (`AddValues`) and subtracting these from the mixture enthalpy (`SubtractValues`). Uncertainties are propagated through the summation and subtraction by the normal means using the uncertainties package.

2.13.6 Solvation Free Energies

Solvation free energies are currently computed using the Yank free energy package using direct molecular simulations. By default the calculations attempt to use 2000 solvent molecules, and the alchemical lambda spacings are selected using the built-in ‘trailblazing’ algorithm.

See the Yank documentation for more details.

2.13.7 Host-Guest Binding Free Energy

Warning: The computation of this property is still in beta. Users are heavily recommended to validate any calculations involving this property.

Host-guest binding free energies are currently computed using the attach-pull-release (APR) method [4] through integration with the pAPRika framework.

2.14 Common Workflows

As may be expected, most of the workflows used to estimate the physical properties within the framework make use of very similar workflows. This page aims to document the built-in ‘template’ workflows from which the more complex physical property estimation workflows are constructed.

2.14.1 Direct Simulation

Properties being estimated using the *direct simulation* calculation layer typically base their workflows off of the `generate_simulation_protocols()` template.

Note: This template currently assumes that a liquid phase property is being computed.

The workflow produced by this template proceeds as follows:

1) 1000 molecules are inserted into a simulation box with an approximate density of 0.95 g / mL using packmol (`BuildCoordinatesPackmol`).

2) the system is parameterized using either the *OpenFF toolkit*, *TLeap* or *LigParGen* depending on the force field being employed (`BuildSmirnoffSystem`, `BuildTLeapSystem` or `BuildLigParGenSystem`).

3) an energy minimization is performed using the default OpenMM energy minimizer (`OpenMMEnergyMinimisation`).

4) the system is equilibrated by running a short NPT simulation for 100000 steps using a timestep of 2 fs and using the OpenMM simulation engine (`OpenMMSimulation`).

5) while the uncertainty in the average observable is greater than the requested tolerance (if specified):

    5a) a longer NPT production simulation is run for 1000000 steps with a timestep of 2 fs and using the OpenMM simulation protocol (`OpenMMSimulation`) with its default Langevin integrator and Monte Carlo barostat.

    5b) the correlated samples are removed from the simulation outputs and the average value of the observable of interest and its uncertainty are computed by bootstrapping with replacement for 250 iterations (`AverageObservable`). See [1] for details of the decorrelation procedure.

    5c) steps 5a) and 5b) are repeated until the uncertainty condition (if applicable) is met.

The decorrelated simulation outputs are then made available ready to be cached by a *storage backend* (`DecorrelateObservables`, `DecorrelateTrajectory`).

2.14.2 MBAR Reweighting

Properties being estimated using the *MBAR reweighting* calculation layer typically base their workflows off of the `generate_reweighting_protocols()` template.

The workflow produced by this template proceeds as follows:

1) for each stored simulation data:

    1a) the cached data is retrieved from disk (`UnpackStoredSimulationData`)

2) the cached data from is concatenated together to form a single trajectory of configurations and observables (`ConcatenateTrajectories`, `ConcatenateStatistics`).

3) for each stored simulation data:

    3a) the system is parameterized using the force field parameters which were used when originally generating the cached data i.e. one of the reference states (`BuildSmirnoffSystem`, `BuildTLeapSystem` or `BuildLigParGenSystem`).

    3b) the reduced potential of each configuration in the concatenated trajectory is evaluated using the parameterized system (`OpenMMEvaluateEnergies`).

4) the system is parameterized using the force field parameters with which the property of interest should be calculated using i.e. of the target state (`BuildSmirnoffSystem`, `BuildTLeapSystem` or `BuildLigParGenSystem`) and the reduced potential of each configuration in the concatenated trajectory is evaluated using the parameterized system (`OpenMMEvaluateEnergies`).

    4a) *(optional)* if the observable of interest is a function of the force field parameters it is recomputed using the target state parameters. These recomputed values then replace the original concatenated observables loaded from the cached data.

5) the reference potentials, target potentials and the joined observables are sub-sampled to only retain equilibrated, uncorrelated samples (`AverageObservable`, `DecorrelateObservables`, `DecorrelateTrajectory`). See [1] for details of the decorrelation procedure.

6) the MBAR method is employed to compute the average value of the observable of interest and its uncertainty at the target state, taking the reference state reduced potentials as input. See [2] for the theory behind this approach. An exception is raised if there are not enough effective samples to reweight (*ReweightObservable*).

In more specialised cases the *generate_base_reweighting_protocols()* template (which *generate_reweighting_protocols()* is built off of) is instead used due to its greater flexibility.

2.14.3 References

2.15 Gradients

A most fundamental feature of this framework is its ability to rapidly compute the gradients of physical properties with respect to the force field parameters used to estimate them.

Note: Prior to v0.3.0 of this framework a combination of re-weighting and the central finite difference was employed to estimate the gradients of observables. From v0.3.0 onwards the fluctuation method [1] is instead used. The change was made to, in future, enable better integration with automatic differentiation libraries such as jax, and differentiable simulation engines such as timemachine.

2.15.1 Theory

The framework currently employs the fluctuation approach [1] to compute gradients of observables with respect to the force field parameters used to estimate them.

This approach may be derived by direct differentiation of the ensemble average an observable X:

$$\langle X\left(\theta\right)\rangle = \frac{1}{Q\left(\theta\right)} \int X\left(\theta\right) \exp\left[-\beta\left(U\left(\vec{r}, V; \theta\right) + pV\right)\right] \mathrm{d}\vec{r}\mathrm{d}V$$

where

$$Q\left(\theta\right) = \int \exp\left[-\beta\left(U\left(\vec{r}, V; \theta\right) + pV\right)\right] \mathrm{d}\vec{r}\mathrm{d}V$$

is the isothermal-isobaric partion function, θ are the force field parameters being used to estimate the observable, U the systems potential energy, $\beta \equiv k_b T$, k_b the Boltzmann constant, T the temperature, p the pressure and V the volume.

The derivative of the ensemble average defined above with respect to a particular force field parameter of interest θ is given by:

$$\frac{\mathrm{d}\langle X\rangle}{\mathrm{d}\theta_i} = \left\langle \frac{\mathrm{d}X}{\mathrm{d}\theta_i} \right\rangle - \beta\, [$$
$$\left\langle X \frac{\mathrm{d}U}{\mathrm{d}\theta_i} \right\rangle - \left\langle \frac{\mathrm{d}U}{\mathrm{d}\theta_i} \right\rangle \langle X\rangle$$

2.15.2 Computing $\mathrm{d}U/\mathrm{d}\theta_i$

While future integrations with differentiable simulation engines such as timemachine will allow $\mathrm{d}U/\mathrm{d}\theta_i$ to be computed directly from molecular simulation runs, currently most common simulation engines do not directly support computing this quantity.

Until such an integration is complete, the framework currently employs a central finite difference approach, whereby

$$\frac{\mathrm{d}U}{\mathrm{d}\theta_i} \approx \frac{U\left(\theta_i + h\right) - U\left(\theta_i - h\right)}{2h}$$

Although more expensive than computing either the forward or backwards derivative, the central difference method should give a more accurate estimate of the gradient at the minima, maxima and transition points. By default a value of $h = \theta_i \times 10^{-4}$ is used. This has been found to yield finite differences which do not suffer from precision issues, while being sufficiently small so as to yield an accurate estimate.

In practice the derivatives obtained by re-evaluating the energies of each configuration in a trajectory generated by a molecular simulation (either after a simulation or after loading one from disk) at each of the perturbed parameters.

While there is an expense associated with extra evaluations of the potential energy function for each configuration, this is mitigated by only computing those terms which depend upon (or may depend upon) θ_i. As an example, when computing derivatives with respect to a bond length the electrostatic and van der Waal contributions are not computed. This significantly speeds up the computation of these derivatives.

The final derivatives are stored in `ObservableArray` objects for convenience and for easy propagation of gradients through workflows. See the *observables documentation* for more information.

2.15.3 References

2.16 Calculation Layers

A `CalculationLayer` is an implementation of one calculation approach for estimating a set of physical properties, such as via molecular simulation or evaluating some QSAR like model.

The framework stacks multiple layers together when estimating a data set of properties.

Fig. 2: A schematic of the layer system. A set of properties to estimate are fed into the first layer. Those which can be calculated are returned back. Those that can't are passed to the next layer until no layer are left.

Each layer will in turn attempt to evaluate the properties being estimated using the specific approach the layer represents, such as by running a set of simulations. If the layer is unable to estimate a given property, for example if a layer does not yet support a given property, or if the layer has insufficient data to reprocesses, the property will be passed to the next layer for it to try and evaluate.

In practice, this allows the framework to attempt to estimate a data set using the most rapid calculation layer first, before moving to successively slower yet more robust layers, and thus enabling as efficient as possible property estimation.

2.16.1 Defining a Calculation Layer

A calculation layer is defined by two objects - a *CalculationLayer* object which implements the main layer logic, and a *CalculationLayerSchema* which defines those settings and options exposed required by the layer.

One *CalculationLayerSchema* will be provided to the for each type of property that the layer is being asked to estimate. The base *CalculationLayerSchema* currently only exposes options for optionally defining either the relative or absolute uncertainty that the layer should attempt to estimate the associated property type to within, however custom schemas can be defined per layer.

The structure of a *CalculationLayer* is relatively simple and permissive:

```
@calculation_layer()
class MyCalculationLayer(CalculationLayer):

    @classmethod
    def required_schema_type(cls):
        return CalculationLayerSchema

    @classmethod
    def _schedule_calculation(
        cls,
        calculation_backend,
        storage_backend,
        layer_directory,
        batch
    ):
        ...
```

The first thing to note is the *calculation_layer()* decorator which is being applied to the class. This registers the calculation layer with the frameworks plug-in system, allowing it to be used in future calculations.

The only other requirements is that the class implement a `required_schema_type` class method, which returns the type of *CalculationLayerSchema* that is associated with this layer, and a `_schedule_calculation()`. The `_schedule_calculation()` is responsible for performing the actual property calculations.

The form of the `_schedule_calculation()` function is very flexible:

```
@classmethod
def _schedule_calculation(
    cls,
    calculation_backend,
    storage_backend,
    layer_directory,
    batch
):

    futures = []

    for queued_property in batch.queued_properties:

        futures.append(
            calculation_backend.submit_task(
                cls.process_property, queued_property, cls.__name__
            )
```

(continues on next page)

(continued from previous page)

```
        )

    return futures
```

It takes as arguments:

- a *CalculationBackend* which is used to asynchronously distribute any calculations across the available compute resources.
- a *StorageBackend* which may be used to store / cache any data generated by the calculations.
- the path to the directory within which all of the calculation working files should be stored.
- the `Batch` of properties which this layer should attempt to estimate. This object includes the properties to estimate, as well as the `CalculationLayerSchema` for each property type.

and must return a list of `Future` objects (which either must be or implement the same API as the asyncio Future object). The easiest way to generate the futures is to perform any calculations using the `calculation_backend` which will automatically return the results of any functions as such.

The future objects returned by `_schedule_calculation()` must return a `CalculationLayerResult` object, which includes

- the estimated property if the calculation was successful (or *`UNDEFINED`* otherwise).
- a list of any exceptions (of type `EvaluatorException`) which were raised during the calculation.
- a list of any data to be stored by the storage backend.

As a minimal example of a method which returns one such object:

```
@classmethod
def process_property(cls, physical_property, **_):
    """Return a result as if the property had been successfully estimated.
    """

    # TODO: Do some calculations

    # Set the property provenance
    physical_property.source = CalculationSource(fidelity=cls.__name__)

    # Return the results object.
    results = CalculationLayerResult()
    results.physical_property = physical_property
    return results
```

2.16.2 Default Schemas

Default schemas for each pair of a calculation layer and a type of physical property may be registered using the *`register_calculation_schema()`* function:

```
# Register the default schema to use for density measurements being estimated
# by the direct simulation calculation layer.
register_calculation_schema(
    property_class=Density,
    layer_class=SimulationLayer,
```

(continues on next page)

(continued from previous page)

```
    schema=Density.default_simulation_schema
)
```

where the schema object should either be an instance of a *CalculationLayerSchema*, or a function with no required arguments which returns a *CalculationLayerSchema*.

A list of the registered schemas is provided by the `registered_calculation_schemas` module attribute.

2.17 Workflow Layers

The *WorkflowCalculationLayer* and *WorkflowCalculationSchema* offer an abstract base implementation for any calculation layers (and their associated schemas) which will perform their calculations using the built-in *workflow engine*.

The *WorkflowCalculationLayer* takes as input from its calculation schema one *WorkflowSchema* object for each type of property to be estimated by this layer. These schemas must *at a minimum* provide both the schemas of the protocols in the workflow, and have the *final_value_source* attribute set to the value of the calculated observable. In addition, the layer fully supports schemas which provide gradient information (see the `gradients_sources` attribute), as well as storing any generated dataclasses (see the *outputs_to_store* attribute) to the available storage backend.

This layer implements three key methods which are available to be overridden by any subclass implementations:

- `_get_workflow_metadata()`: a method which returns the dictionary of *metadata* which will be made available to the workflow (see the *default metadata* section for details).
- `_build_workflow_graph()`: the method which will construct the *workflow graph* to execute using the input workflow schemas and the metadata generated by the layer.
- *workflow_to_layer_result()*: a method which will map any *WorkflowResult* objects generated by the workflow graph into the *CalculationLayerResult* objects which the layer requires.

The workflow layer will by default tag each property estimated using it (or one of its derivatives) with a *CalculationSource* with the *fidelity* attribute set to the name of the layer, and the *provenance* attribute set to the schema of the workflow used to generate the property.

2.17.1 Default Metadata

The metadata provided to the workflows generated by this layer is generated on a per property to estimate basis mainly using the *generate_default_metadata()* function. It includes:

Key	Type	Description
thermodynamic_state	*ThermodynamicState*	The state at which the to perform any calculations .
substance	*Substance*	The substance to use in any calculations.
components	[*Substance*]	The components present in the main substance.
target_uncertainty	Quantity	The target uncertainty of any calculations defined by the calculation schema.
per_component_uncertainty	Quantity	The target_uncertainty divided by sqrt(substance.n_components + 1)
force_field_path	str	A file path to the force field parameters to use.
parameter_gradient_keys	[*ParameterGradientKey*]	The parameters to differentiate any observables with respect to (if any).

2.18 The Direct Simulation Layer

The *SimulationLayer* is a calculation layer which employs molecular simulation to estimate data sets of physical properties. It inherits the *WorkflowCalculationLayer* base layer, and primarily makes use of the built-in *workflow* engine to perform the required calculations.

The simulation layer is expected to *almost always* be able to estimate any properties requested of it (with exceptions being where a workflow schema has not yet been defined for a class of properties, or where an unexpected error occurs), and can be thought of as a safe 'fallback' layer when no other calculation approach are able to estimate particular properties.

It is expected that *workflow schemas* passed to the simulation layer should be able to estimate the gradients of the observable they aim to calculate, as well as specify a set of :doc:` storage/dataclasses <storage/dataclasses>` which contain the data generated by the molecular simulations.

2.18.1 Default Metadata

The simulation layer makes the same set of metadata available to its workflows as the *parent workflow layer*.

2.19 The MBAR Reweighting Layer

The *ReweightingLayer* is a calculation layer which employs the Multistate Bennett Acceptance Ratio (MBAR) method to calculate observables at states which have not been previously simulated, but for which simulations have been previously run at similar states and their data cached. It inherits the *WorkflowCalculationLayer* base layer, and primarily makes use of the built-in *workflow* engine to perform the required calculations.

Because MBAR is a technique which reprocesses exisiting simulation data rather than re-running new simulations, it is typically several fold faster than the *simulation layer* provided it has cached simulation data (made accessible via a *storage backend*) available. Any properties for which the required data (see *Calculation Schema*) is not available will be skipped.

2.19.1 Theory

The theory behind applying MBAR to reweighting observables from a simulated state to an unsimulated state is covered in detail in the publication Configuration-Sampling-Based Surrogate Models for Rapid Parameterization of Non-Bonded Interactions.

2.19.2 Calculation Schema

The reweighting layer will be provided with one *ReweightingSchema* per type of property that it is being requested to estimate. It builds off of the base *WorkflowCalculationSchema* schema providing an additional *storage_queries* attribute.

The *storage_queries* attribute will contain a dictionary of *SimulationDataQuery* which will be used by the layer to access the data required for each property from the storage backend. Each key in this dictionary will correspond to the key of a piece of metadata made available to the property workflows.

2.19.3 Default Metadata

The reweighting layer makes available the default metadata provided by the *parent workflow layer* in addition to any cached data retrieved via the schemas `storage_queries`.

When building the metadata for each property, a copy of the query will be made and any of the supported attributes (currently only `substance`) whose values are set as `PlaceholderValue` objects will have their values updated using values directly from the property. This query will then be passed to the storage backend to retrieve any matching data.

The matching data will be stored as a list of tuples of the form:

```
(object_path, data_directory, force_field_path)
```

where `object_path` is the file path to the stored dataclass, the `data_directory` is the file path to the ancillary data directory and `force_field_path` is the file path to the force field parameters which were used to generate the data originally.

This list of tuples will be made available as metadata under the key that was associated with the query.

2.20 Workflows

The framework offers a lightweight workflow engine for executing graphs of tasks using the available *calculation backends*. While lightweight, it offers a large amount of extensibility and flexibility, and is currently used by both the *simulation* and *reweighting* layers to perform their required calculations.

A workflow is a wrapper around a collection of tasks that should be executed in succession, and whose outputs should be made available as the input to others.

Fig. 3: A an example workflow which combines a protocol which will build a set of coordinates for a particular system, assign parameters to that system, and then perform an energy minimisation.

The workflow engine offers a number of advanced features such as the *automatic reduction of redundant tasks*, and *looping over parts of a workflow*

2.20.1 Building Workflows

At its core a workflow must define the tasks which need to be executed, and where the inputs to those tasks should be sourced from. Each task to be executed is represented by a *protocol object*, with each protocol requiring a specific set of user specified inputs:

```python
# Define a protocol which will build some coordinates for a system.
build_coordinates = BuildCoordinatesPackmol("build_coordinates")
build_coordinates.max_molecules = 1000
build_coordinates.mass_density = 1.0 * unit.gram / unit.millilitre
build_coordinates.substance = Substance.from_components("O", "CO")

# Define a protocol which will assign force field parameters to the system.
assign_parameters = BuildSmirnoffSystem(f"assign_parameters")
assign_parameters.water_model = BuildSmirnoffSystem.WaterModel.TIP3P
assign_parameters.force_field_path = "openff-1.0.0.offxml"

# Set the `coordinate_file_path` input of the `assign_parameters` protocol
```

(continues on next page)

(continued from previous page)

```
# to the `coordinate_file_path` output of the `build_coordinates` protocol.
assign_parameters.coordinate_file_path = ProtocolPath(
    "coordinate_file_path", build_coordinates.id
)
```

The *ProtocolPath* object is used to reference the output of another protocol in the workflow, and will be replaced by the value of that output once that protocol has been executed by the workflow engine. It is constructed from two parts:

- the name of the output attribute to reference.
- the unique id of the protocol to take the output from.

To turn these tasks into a valid workflow which can be automatically executed, they must first be converted to a *workflow schema*:

```
# Create the schema object.
schema = WorkflowSchema()
# Add the individual protocol's schema representations to the workflow schema.
schema.protocol_schemas = [build_coordinates.schema, assign_parameters.schema]

# Create the executable workflow object from its schema.
workflow = Workflow.from_schema(schema, metadata=None)
```

A *Workflow* may either be synchronously executed in place yielding a *WorkflowResult* object directly:

```
workflow_result = workflow.execute()
```

or asynchronously using a calculation backend yielding a `Future` like object which will eventually return a *WorkflowResult*:

```
with DaskLocalCluster() as calculation_backend:
    result_future = workflow.execute(calculation_backend=calculation_backend)
```

In addition, a workflow may be add to, and executed as part as a larger *workflow graphs*.

2.20.2 Workflow Schemas

A *WorkflowSchema* is a blueprint from which all *Workflow* objects are constructed. It will predominantly define the tasks which compose the workflow, but may optionally define:

- *final_value_source*: A reference to the protocol output which corresponds to the value of the main observable calculated by the workflow.
- `gradients_sources`: A list of references to the protocol outputs which correspond to the gradients of the main observable with respect to a set of force field parameters.
- *outputs_to_store*: A list of *data classes* whose values will be populated from protocol outputs.
- *protocol_replicators*: A set of *replicators* which are used to flag parts of a workflow which should be replicated.

Each of these attributes will control whether the *value*, *gradients* and *data_to_store* attributes of the *WorkflowResult* results object will be populated respectively when executing a workflow.

Metadata

Because a schema is purely a blueprint for a general workflow, it need not define the exact values of all of the inputs of its constituent tasks. Consider the above example workflow for constructing a set of coordinates and assigning force field parameters to them. Ideally this one schema could be reused for multiple substances. This is made possible through a workflows *metadata*.

Each protocol within a workflow may access a dictionary of values unique to that workflow (termed here *metadata*) which is defined when the `Workflow` object is created from its schema.

This metadata may be accessed by protocols via a fictitious `"global"` protocol whose outputs map to the `metadata` dictionary:

```python
build_coordinates = BuildCoordinatesPackmol("build_coordinates")
build_coordinates.substance = ProtocolPath("substance", "global")

# ...

substances = [
    Substance.from_components("CO"),
    Substance.from_components("CCO"),
    Substance.from_components("CCCO"),
]

for substance in substances:

    # Define the metadata to make available to the workflow protocols.
    metadata = {"substance": substance}
    # Create the executable workflow object from its schema.
    workflow = Workflow.from_schema(schema, metadata=metadata)

    # Execute the workflow ...
```

the created workflow will contain the `build_coordinates` protocol but with its `substance` input set to the value from the `metadata` dictionary.

2.21 Replicators

A *`ProtocolReplicator`* is the workflow equivalent of a `for` loop. It is statically evaluated when a *`Workflow`* is created from its schema. This is useful when parts of a workflow should be run multiple times but using different values for certain protocol inputs.

Note: The syntax of replicators is still rather rough around the edges, and will be refined in future versions of the framework.

Each *`ProtocolReplicator`* requires both a unique id and the set of *template values* which the replicator will 'loop' over to be defined. These values must either be a list of constant values or a reference to a list of values provided as *metadata*.

The 'loop variable' is referenced by protocols in the workflow using the *`ReplicatorValue`* placeholder input, where the value is linked to the replicator through the replicators unique id.

As an example, consider the case where a set of coordinates should be built for each component in a substance:

```python
# Create the replicator object, and assign it a unique id.
replicator = ProtocolReplicator(replicator_id="component_replicator")
# Instruct the replicator to loop over all of the components of the substance
# made available by the global metadata
replicator.template_values = ProtocolPath("substance.components", "global")

# Define a protocol which will build some coordinates for a system.
build_coords = BuildCoordinatesPackmol("build_coords_" + replicator.placeholder_id})
# Instruct the protocol to use the value specified by the replicator.
build_coords.substance = ReplicatorValue(replicator.id)

# Build the schema containing the protocol and the replicator
schema = WorkflowSchema()
schema.protocol_schemas = [build_coords.schema]
schema.protocol_replicators = [replicator]
```

The requirement for a protocol to be replicated by a replicator is that its id *must* contain the replicators `placeholder_id` - this is a simple string which the workflow engine looks for when applying the replicator. The contents of this schema can be easily inspected by printing its JSON representation:

```json
{
    "@type": "openff.evaluator.workflow.schemas.WorkflowSchema",
    "protocol_replicators": [
        {
            "@type": "openff.evaluator.workflow.schemas.ProtocolReplicator",
            "id": "component_replicator",
            "template_values": {
                "@type": "openff.evaluator.workflow.utils.ProtocolPath",
                "full_path": "global.substance.components"
            }
        }
    ],
    "protocol_schemas": [
        {
            "@type": "openff.evaluator.workflow.schemas.ProtocolSchema",
            "id": "build_coords_$(component_replicator)",
            "inputs": {
                ".substance": {
                    "@type": "openff.evaluator.workflow.utils.ReplicatorValue",
                    "replicator_id": "component_replicator"
                }
            },
            "type": "BuildCoordinatesPackmol"
        }
    ]
}
```

It can be clearly seen that the schema only contains a single protocol entry, with the placeholder id present in its unique id. Once a workflow is created from this schema however:

```python
# Define some metadata
metadata = {"substance": Substance.from_components("O", "CO")}
```

(continues on next page)

(continued from previous page)

```
# Build the workflow from the schema.
workflow = Workflow.from_schema(schema, metadata)
# Output the contents of the workflow as JSON.
print(workflow.schema.json())
```

it can be seen that the replicator has been correctly been applied and the workflow now contains one protocol for each component in the substance passed as metadata:

```
{
    "@type": "openff.evaluator.workflow.schemas.WorkflowSchema",
    "protocol_schemas": [
        {
            "@type": "openff.evaluator.workflow.schemas.ProtocolSchema",
            "id": "build_coords_0",
            "inputs": {
                ".substance": {
                    "@type": "openff.evaluator.substances.components.Component",
                    "smiles": "O"
                }
            },
            "type": "BuildCoordinatesPackmol"
        },
        {
            "@type": "openff.evaluator.workflow.schemas.ProtocolSchema",
            "id": "build_coords_1",
            "inputs": {
                ".substance": {
                    "@type": "openff.evaluator.substances.components.Component",
                    "smiles": "CO"
                }
            },
            "type": "BuildCoordinatesPackmol"
        }
    ]
}
```

In both cases the replicators `placeholder_id` has been replaced with the index of the value it was replicated for, and the substance input has been correctly set to the actual array value.

2.21.1 Nested Replicators

Replicators can be applied to other replicators to achieve a result similar to a set of nested for loops. For example the below loop:

```
components = [Component("O"), Component("CO")]
n_mols = [[1000], [500]]

for i, component in enumerate(components):

    for component_n_mols in n_mols[i]:

        ...
```

can readily be reproduced using replicators:

```python
# Define a replicator which will loop over all components in the substance.
component_replicator = ProtocolReplicator(replicator_id="components")
component_replicator.template_values = ProtocolPath("components", "global")

# Define a replicator to loop over the number of each component to add.
n_mols_replicator_id = f"n_mols_{component_replicator.placeholder_id}"

n_mols_replicator = ProtocolReplicator(replicator_id=n_mols_replicator_id)
n_mols_replicator.template_values = ProtocolPath(
    f"n_mols[{component_replicator.placeholder_id}]", "global"
)

# Define the suffix which must be applied to protocols to be replicated
id_suffix = f"{component_replicator.placeholder_id}_{n_mols_replicator.placeholder_id}"

# Define a protocol which will build some coordinates for a system.
build_coordinates = BuildCoordinatesPackmol(f"build_coordinates_{id_suffix}")
build_coordinates.substance = ReplicatorValue(component_replicator.id)
build_coordinates.max_molecules = ReplicatorValue(n_mols_replicator.id)

# Build the schema containing the protocol and the replicator
schema = WorkflowSchema()
schema.protocol_schemas = [build_coordinates.schema]
schema.protocol_replicators = [component_replicator, n_mols_replicator]

# Define some metadata
metadata = {
    "components": [Component("O"), Component("CO")],
    "n_mols": [[1000], [500]]
}

# Build the workflow from the created schema.
workflow = Workflow.from_schema(schema, metadata)
# Print the JSON representation of the workflow.
print(workflow.schema.json(format=True))
```

Here the `component_replicator` placeholder id has been appended to the `n_mols_replicator` id to inform the workflow engine that the later is a child of the former. The `component_replicator` placeholder id is then used as an index into the `n_mols` array. This results in the following schema as desired:

```json
{
    "@type": "openff.evaluator.workflow.schemas.WorkflowSchema",
    "protocol_schemas": [
        {
            "@type": "openff.evaluator.workflow.schemas.ProtocolSchema",
            "id": "build_coordinates_0_0",
            "inputs": {
                ".max_molecules": 1000,
                ".substance": {
                    "@type": "openff.evaluator.substances.components.Component",
                    "smiles": "O"
                }
```

(continues on next page)

(continued from previous page)

```
            },
            "type": "BuildCoordinatesPackmol"
        },
        {
            "@type": "openff.evaluator.workflow.schemas.ProtocolSchema",
            "id": "build_coordinates_1_0",
            "inputs": {
                ".max_molecules": 500,
                ".substance": {
                    "@type": "openff.evaluator.substances.components.Component",
                    "smiles": "CO"
                }
            },
            "type": "BuildCoordinatesPackmol"
        }
    ]
}
```

2.22 Workflow Graphs

A *`WorkflowGraph`* is a collection of *`Workflow`* objects which should be executed together. The primary advantage of executing workflows via the graph object is that the graph will automatically take advantage of the *protocols* built in redundancy / merging support to collapse duplicate tasks across multiple workflows.

As an example, consider the case of executing workflows to estimate the density and the dielectric constant at the same state point, for the same substance, and using the same force field parameters:

```
density_schema = Density.default_simulation_schema()
dielectric_schema = DielectricConstant.default_simulation_schema()

density_workflow = Workflow.from_schema(density_schema, metadata)
dielectric_workflow = Workflow.from_schema(dielectric_schema, metadata)

print(len(density_workflow.protocols), len(dielectric_workflow.protocols))

workflow_graph = WorkflowGraph()
workflow_graph.add_workflows(density_workflow, dielectric_workflow)

print(len(workflow_graph.protocols))
```

The final workflow graph has roughly half the total number of density and dielectric protocols to be executed. This is expected as both the density and dielectric workflows are almost identical, except for the final analysis steps.

Graphs can be executed either in place without using a calculation backend in the same way that *workflows can*.

2.23 Protocols

The *Protocol* class represents a single task to be executed, whether that be as a standalone task or as a task which is part of some larger workflow. The task encoded by a protocol may be as simple as adding two numbers together or even as complex as performing entire free energy simulations:

```python
from openff.evaluator.protocols.miscellaneous import AddValues

# Create the protocol and assign it some unique name.
add_numbers = AddValues(protocol_id="add_values")
# Set the numbers to add together
add_numbers.values = [1, 2, 3, 4]

# Execute the protocol
add_numbers.execute()

# Retrieve the output
result = add_numbers.result
```

2.23.1 Inputs and Outputs

Each protocol exposes a set of the required inputs as well as the produced outputs. These inputs may either be set as a constant directly, or if used as part of a *workflow*, can take their value from one of the outputs of another protocol.

Fig. 4: A selection of the inputs and outputs of the *OpenMMSimulation* protocol.

A surprisingly rich spectrum of workflows can be constructed by chaining together many relatively simple protocols.

The inputs and outputs of a protocol are defined using the custom *InputAttribute* and *OutputAttribute* descriptors:

```python
class AddValues(Protocol):

    # Define the inputs that the protocol requires
    values = InputAttribute(
        docstring="The values to add together.",
        type_hint=list, default_value=UNDEFINED
    )

    # Define the outputs that the protocol will produce
    # once it is executed.
    result = OutputAttribute(
        docstring="The sum of the values.",
        type_hint=typing.Union[int, float, unit.Measurement, unit.Quantity],
    )

    def _execute(self, directory, available_resources):
        ...

    def validate(self, attribute_type=None):
        ...
```

Here we have defined a `values` input to the protocol and a `result` output. Both descriptors require a `docstring` and a `type_hint` to be provided.

The `type_hint` will be used by the workflow engine to ensure that a protocol which takes its input as the output of another protocol is receiving values of the correct type. Currently the `type_hint` can be any type of python class, or a `Union` of multiple types should the protocol allow for that.

In addition, the input attribute must specify a `default_value` for the attribute. This can either be a constant value, or a value set by some function such as a `lambda` statement:

```
some_input = InputAttribute(
    docstring="Takes it's default value from a function.",
    type_hint=int,
    default_value=lambda: return 1 + 1
)
```

In the above example we set the default value of `values` to *`UNDEFINED`* in order to specify that this input must be set by the user. The custom *`UNDEFINED`* class is used in place of `None` as `None` may be a valid input value for some attributes.

2.23.2 Task Execution

In addition to defining its inputs and outputs, a protocol must also implement an `_execute()` function which handles the main logic of the task:

```
def _execute(self, directory, available_resources):

    self.result = self.values[0]

    for value in self.values[1:]:
        self.result += value
```

The function is passed the directory in which it should run and create any working files, as well as a *`ComputeResources`* object which describes which compute resources are available to run on. This function *must* set all of the output attributes of the protocol before returning.

The private `_execute()` function which must be implemented should not be confused with the public *`execute()`* function. The public *`execute()`* function implements some common protocol logic (such as validating the inputs and creating the directory to run in if needed) before calling the private `_execute()` function.

2.23.3 Protocol Validation

The protocols inputs will automatically be validated before `_execute()` is called - this validation includes making sure that all of the non-optional inputs have been set, as well as ensuring they have been set to a value of the correct type. Protocols may implement additional validation logic by implementing a *`execute()`* function:

```
def validate(self, attribute_type=None):

    super(AddValues, self).validate(attribute_type)

    if len(self.values) < 1:
        raise ValueError("There were no values to add together")
```

2.23.4 Schemas

Every protocol has a *ProtocolSchema* representation which uniquely describes the protocol, and from which the protocol can be exactly recreated. The schema stores not only the type of protocol which it represents, but also the values of each of the inputs. Protocol schemas are fully JSON serializable. The following is an example schema for the above `add_numbers` protocol:

```
{
  "@type": "openff.evaluator.workflow.schemas.ProtocolSchema",
  "id": "add_values",
  "inputs": {
    ".allow_merging": true,
    ".values": [1, 2, 3, 4]
  },
  "type": "AddValues"
}
```

A protocols schema can be accessed via it's *schema* attribute. A protocol can be directly created from its schema representation by calling the schema's *to_protocol()* function.

2.23.5 Merging Protocols

When executing multiple workflows together (e.g. executing a workflow to estimate a substances density and potential energy) there is a large likelihood that some of tasks in those two workflows will be identical. Examples may include two workflows requiring protocols which build a set of coordinates, or assigning the same set of parameters to those coordinates.

Protocols have built-in support for comparing whether they are performing the same task / calculation as another protocol through the *can_merge()* and *merge()* functions:

- The *can_merge()* function checks to see whether two protocols are performing an identical task and hence whether they should be merged or not.
- The *merge()* function handles the actual merging of two protocols which can be merged.

The default *can_merge()* function takes advantage of the `merge_behvaiour` attribute of the different input descriptors. The `merge_behvaiour` attribute describes how each input should be considered when checking to see if two protocols can be merged:

```
max_molecules = InputAttribute(
    docstring="The maximum number of molecules to be added to the system.",
    type_hint=int,
    default_value=1000,
    merge_behavior=MergeBehaviour.ExactlyEqual
)
```

The most common behavior is to require that the inputs must be `ExactlyEqual` in order for two protocols two be considered to be identical. However, for some inputs such as the timestep of a simulation or the number of steps to simulate for, the exact values of the inputs don't necessarily need to be equal but rather, we may just wish to take the larger / smaller of the two inputs:

```
timestep = InputAttribute(
    docstring="The timestep to evolve the system by at each step.",
    type_hint=unit.Quantity,
    merge_behavior=InequalityMergeBehaviour.SmallestValue,
```

(continues on next page)

(continued from previous page)

```
    default_value=2.0 * unit.femtosecond,
)

total_number_of_iterations = InputAttribute(
    docstring="The number of times to propogate the system forward by.",
    type_hint=int,
    merge_behavior=InequalityMergeBehaviour.LargestValue,
    default_value=1,
)
```

This can be accomplished using the *InequalityMergeBehaviour* enum.

The default *merge()* function also relies upon the `merge_behaviour` attributes to determine which values of the inputs should be retained when merging two protocols.

2.24 Protocol Groups

The *ProtocolGroup* class represents a collection of *protocols* which have been grouped together. All protocols within a group will be executed together on a single compute resources, i.e. there is currently no support for executing protocols within a group in parallel.

Protocol groups have a specialised *ProtocolGroupSchema* which is essentially a collection of *ProtocolSchema* objects.

2.24.1 Conditional Protocol Groups

A *ConditionalGroup* is a special class of *ProtocolGroup* which will execute all of the grouped protocols again and again until a set of conditions has been met or until a maximum number of iterations (see *max_iterations*) has been performed. They can be thought of as being a protocol representation of a `while` statement.

Each condition to be met is represented by a *Condition* object:

```
condition = ConditionalGroup.Condition()

# Set the left and right hand values.
condition.left_hand_value = ...
condition.right_hand_value = ...

# Choose the type of condition
condition.type = ConditionalGroup.Condition.Type.LessThan
```

The left and right hand values can either be constants, or come from the output of another protocol (including grouped protocols) using a *ProtocolPath*. Currently a condition can either check that a value is less than or greater than another value.

Conditional groups expose a *current_iteration* attribute which tracks how many times the grouped protocols have been executed. This can be used as input by any of the grouped protocols and is useful, for example, to run a simulation for longer and longer until the groups condition has been met:

```
conditional_group = ConditionalGroup("conditional_group")

# Set up protocols to run a simulation and then to extract the
```

(continues on next page)

(continued from previous page)

```
# value of the density and its uncertainty.
simulation = OpenMMSimulation("simulation")
simulation.input_coordinate_file = "coords.pdb"
simulation.parameterized_system = ...

extract_density = AverageObservable("extract_density")
extract_density.observable = simulation.observables["Density"]

# Set the total number of iterations the simulation should perform to be equal
# to the current iteration of the group. I.e the simulation should perform a
# new iteration at each group iteration.
simulation.total_number_of_iterations = ProtocolPath(
    "current_iteration", conditional_group.id
)

# Add the protocols to the group.
conditional_group.add_protocols(production_simulation, analysis_protocol)

# Set up a condition which will check if the uncertainty is less than
# some threshold.
condition = ConditionalGroup.Condition()
condition.condition_type = groups.ConditionalGroup.Condition.Type.LessThan

condition.right_hand_value = 0.5 * unit.gram / unit.millilitre
condition.left_hand_value = ProtocolPath(
    "value.error", conditional_group.id, analysis_protocol.id
)

# Add the condition.
conditional_group.add_condition(condition)
```

It is this idea which is used to continue running a molecular simulations until an observable of interest (such as the density) has been calculated to within a specified uncertainty.

2.25 Observables

A key feature of this framework is its ability to compute the gradients of physical properties with respect to the force field parameters used to estimate them. This requires the framework be able to, internally, be able to not only track the gradients of all quantities which combine to yield the final observable of interest, but to also be able to propagate the gradients of those composite quantities through to the final value.

The framework offers three such objects to this end (`Observable`, `ObservableArray` and `ObservableFrame` objects) which will be covered in this document.

Note: In future versions of the framework the objects described here will likely be at least in part deprecated in favour of using full automatic differentiation libraries such as jax. Supporting these libraries will take a large re-write of the framework however, as well as full support between differentiable simulation engines like timemachine and the OpenFF toolkit. As such, these objects are implemented as stepping stones which can be gently phased out while working towards that larger, more modern goal.

2.25.1 Observable Objects

The base object used to track observables is the `Observable` object. It stores the average value, the standard error in the value and the gradient of the value with respect to force field parameters of interest.

Currently the value and error are internally stored in a composite `Measurement` object, which themselves wrap around the uncertainties package. This allows uncertainties to be automatically propagated through operations without the need for user intervention.

Note: Although uncertainties are automatically propagated, it is still up to property estimation workflow authors to ensure that such propagation (assuming a Gaussian error model) is appropriate. An alternative, which is employed throughout the framework is to make use of the bootstrapping technique.

Gradients are stored in a list as `ParameterGradient` gradient objects, which store both the floating value of the gradient alongside an identifying `ParameterGradientKey`.

Supported Operations

- **+ and -**: `Observable` objects can be summed with and subtracted from other `Observable` objects, `Quantity` objects, floats or integers. When two `Observable` objects are summed / subtracted, their gradients are combined by summing / subtracting also. When an `Observable` is summed / subtracted with a `Quantity`, `float` or `int` object it is assumed that these objects do not depend on any force field parameters.
- *****: `Observable` objects may be multiplied by other `Observable` objects, `Quantity` objects, and `float` or `int` objects. When two `Observable` objects are multiplied their gradients are propagated using the product rule. When an `Observable` is multiplied by a `Quantity`, `float` or `int` object it is assumed that these objects do not depend on any force field parameters.
- **/**: `Observable` objects may be divided by other `Observable` objects, `Quantity` objects, and `float` or `int` objects. Gradients are propagated through the division using the quotient rule. When an `Observable` is divided by a `Quantity`, `float` or `int` object (or when these objects are divided by an `Observable` object) it is assumed that these objects do not depend on any force field parameters.

In all cases two `Observable` objects can only be operated on provided the contain gradient information with respect to the same set of force field parameters.

2.25.2 Observable Arrays

An extension of the `Observable` object is the `ObservableArray` object. Unlike an `Observable`, an `ObservableArray` object does not contain error information, but rather the value it stores and the gradients of that value should be a numpy array with `shape=(n_data_points, n_dimensions)`. It is designed to store information such as the potential energy evaluated at each configuration sampled during a simulation, as well as the gradient of the potential, which can then be ensemble averaged using a fluctuation formula to propagate the gradients through to the average.

Like with `Observable` objects, gradients are stored in a list as `ParameterGradient` gradient objects. The length of the gradients is required to match the length of the value array.

`ObservableArray` objects may be concatenated together using their `join()` method or sub-sampled using their `subset()` method.

Supported Operations

The *ObservableArray* object supports the same operations as the *Observable* object, whereby all operations are applied elementwise to the stored arrays.

2.25.3 Observable Frames

An *ObservableFrame* is a wrapper around a collection of *ObservableArray* which contain the types of observable specified by the *ObservableType* enum. It behaves as a dictionary which can take either an *ObservableType* or a string value of an *ObservableType* as an index.

Like an *ObservableArray*, observable frames may be concatenated together using their *join()* method or sub-sampled using their *subset()* method.

Supported Operations

No operations are supported between observable frames.

submit_task

2.26 Calculation Backends

A *CalculationBackend* is an object used to distribute calculation tasks across available compute resources. This is possible through specific backends which integrate with libraries such as multiprocessing, dask, parsl and cerlery.

Each backend is responsible for creating *compute workers*. A compute worker is an entity which has a set amount of dedicated compute resources available to it and which can execute python functions using those resources. Calculation backends may spawn multiple workers such that many tasks and calculations can be performed simultaneously.

A compute worker can be as simple as a new multiprocessing `Process` or something more complex like a dask worker. The resources available to a worker are described by the *ComputeResources* object.

CalculationBackend classes have a relatively simple structure:

```
class MyCalculationBackend(CalculationBackend):

    def __init__(self, number_of_workers, resources_per_worker):
        ...

    def start(self):
        ...

    def stop(self):
        ...

    def submit_task(self, function, *args, **kwargs):
        ...
```

By default they implement a constructor which takes as input the number of workers that the backend should initially spawn as well as the compute resources which are available to each. They must further implement:

- a *start()* method which spawns the initial set of compute workers.

- a *stop()* method which should kill all workers spawned by the backend as well as cleanup any temporary worker files.
- a *submit_task()* method which takes a function to be execute by a worker, and a set of `args` and `kwargs` to pass to that function.

The *submit_task()* must run asynchronously and return an asyncio `Future` object (or an object which implements the same API) when called, which can then be queried for when the task has completed.

All calculation backends are implemented as context managers such that they can be used as:

```
with MyCalculationBackend(number_of_workers=..., resources_per_worker...) as backend:
    backend.submit_task(...)
```

where the *start()* and *stop()* methods will be called automatically.

2.27 Dask Backends

The framework implements a number of calculation backends which integrate with the `dask` distributed and job-queue libraries.

2.27.1 Dask Local Cluster

The *DaskLocalCluster* backend wraps around the dask LocalCluster class to distribute tasks on a single machine:

```
worker_resources = ComputeResources(
    number_of_threads=1,
    number_of_gpus=1,
    preferred_gpu_toolkit=GPUToolkit.CUDA,
)

with DaskLocalCluster(number_of_workers=1, resources_per_worker=worker_resources) as␣
↪local_backend:
    local_backend.submit_task(logging.info, "Hello World")
    ...
```

Its main purpose is for use when debugging calculations locally, or when running calculations on machines with large numbers of CPUs or GPUs.

2.27.2 Dask HPC Cluster

The *DaskLSFBackend* and *DaskPBSBackend* backends wrap around the dask LSFCluster and PBSCluster classes respectively, and both inherit the *BaseDaskJobQueueBackend* class which implements the core of their functionality. They predominantly run in an adaptive mode, whereby the backend will automatically scale up or down the number of workers based on the current number of tasks that the backend is trying to execute.

These backends integrate with the queueing systems which most HPC cluster use to manage task execution. They work by submitting jobs into the queueing system which themselves spawn dask workers, which in turn then execute tasks on the available compute nodes:

```
# Create the object which describes the compute resources each worker should request from
# the queueing system.
```

(continues on next page)

(continued from previous page)

```python
worker_resources = QueueWorkerResources(
    number_of_threads=1,
    number_of_gpus=1,
    preferred_gpu_toolkit=QueueWorkerResources.GPUToolkit.CUDA,
    per_thread_memory_limit=worker_memory,
    wallclock_time_limit="05:59",
)

# Create the backend object.
setup_script_commands = [
    f"conda activate evaluator",
    f"module load cuda/10.1",
]

calculation_backend = DaskLSFBackend(
    minimum_number_of_workers=1,
    maximum_number_of_workers=max_number_of_workers,
    resources_per_worker=queue_resources,
    queue_name="gpuqueue",
    setup_script_commands=setup_script_commands,
)

# Perform some tasks.
with calculation_backend:
    calculation_backend.submit_task(logging.info, "Hello World")
    ...
```

The `setup_script_commands` argument takes a list of commands which should be run by the queue job submission script before spawning the actual worker. This enables setting up custom environments, and setting any required environmental variables.

Configuration

To ensure optimal behaviour we recommend changing / uncommenting the following settings in the dask distributed configuration file (this can be found at `~/.config/dask/distributed.yaml`):

```yaml
distributed:

    worker:
        daemon: False

    comm:
        timeouts:
            connect: 10s
            tcp: 30s

    deploy:
        lost-worker-timeout: 15s
```

See the dask documentation for more information about changing dask settings.

2.28 Storage Backends

A *StorageBackend* is an object used to store data generated as part of property calculations, and to retrieve that data for use in future calculations.

In general, most data stored in a storage backend is stored in two parts:

- A JSON serialized representation of this class (or a subclass), which contains lightweight information such as the state and composition of a system.
- 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, such as simulation trajectories, whose files are referenced by their file name by the data object.

The ancillary directory-like structure is not required if the data may be suitably stored in the data object itself.

2.28.1 Data Storage / Retrieval

Each piece of data which is stored in a backend must inherit from the *BaseStoredData* class, will be assigned a unique key. This unique key is both useful for tracking provenance if this data is re-used in future calculations, and also can be used to retrieve the piece of data from the storage system.

In addition to retrieval using the data keys, each backend offers the ability to perform a 'query' to retrieve data which matches a set of given criteria. Data queries are implemented via *BaseDataQuery* objects, which expose different options for querying for specific types of data (such a simulation data, trained models, etc.).

A query may be used for example to match all simulation data that was generated for a given *Substance* in a particular phase:

```python
# Look for all simulation data generated for liquid water
substance_query = SimulationDataQuery()

substance_query.substance = Substance.from_components("O")
substance_query.property_phase = PropertyPhase.Liquid

found_data = backend.query(substance_query)
```

The returned `found_data` will be a dictionary with keys of tuples and values as lists of tuples. Each key will be a tuple of the values which were matched, for example the matched thermodynamic state, or the matched substance. For each value tuple in the tuple list, the first item in the tuple is the unique key of the found data object, the second item is the data object itself, and the final object is the file path to the ancillary data directory (or `None` if none is present).

See the *Data Classes and Queries* page for more information about the available data classes, queries and their details.

2.28.2 Implementation

A *StorageBackend* must at minimum implement a structure of:

```python
class MyStorageBackend(StorageBackend):

    def _store_object(self, object_to_store, storage_key=None, ancillary_data_path=None):
        ...

    def _retrieve_object(self, storage_key, expected_type=None):
        ...
```

(continues on next page)

(continued from previous page)

```
    def _object_exists(self, storage_key):
        ...
```

where

- `_store_object()` must store a *BaseStoredData* object as well as optionally its ancillary data directory, and return a unique key assigned to that object.
- `_retrieve_object()` must return the *BaseStoredData* object which has been assigned a given key if the object exists in the system, as well as the file path to ancillary data directory if it exists.
- `_object_exists()` should return whether any object still exists in the storage system with a given key.

All of these methods will be called under a reentrant thread lock and may be considered as thread safe.

2.29 Data Classes and Queries

All data which is to be stored within a *StorageBackend* must inherit from the *BaseStoredData* class. More broadly there are typically two types of data which are expected to be stored:

- *HashableStoredData* - data which is readily hashable and can be quickly queried for in a storage backend. The prime examples of such data are *ForceFieldData*, whose hash can be easily computed from the file representation of a force field.
- *ReplaceableData* - data which should be replaced in a storage backend when new data of the same type, but which has a higher information content, is stored in the backend. An example of this is when storing a piece of *StoredSimulationData* in the backend which was generated for a particular *Substance* and at the same *ThermodynamicState* as an existing piece of data, but which stores many more uncorrelated configurations.

Every data class **must** be paired with a corresponding data query class which inherits from the *BaseDataQuery* class. In addition, each data object must implement a *to_storage_query()* function which returns the data query which would uniquely match that data object. The *to_storage_query()* is used heavily by storage backends when checking if a piece of data already exists within the backend.

2.29.1 Force Field Data

The *ForceFieldData* class is used to *ForceFieldSource* objects within the storage backend. It is a hashable storage object which allows for rapidly checking whether any calculations have been previously been performed for a particular force field source.

It has a corresponding *ForceFieldQuery* class which can be used to query for particular force field sources within a storage backend.

2.29.2 Cached Simulation Data

Classes derived from the *BaseSimulationData* class are used to store the data generated by molecular simulation. The data object primarily records the *Substance*, *PropertyPhase* and *ThermodynamicState* that the simulation was run at, as well as provenance about the calculation and the force field parameters used (as the key of the force field in the storage system).

It has a corresponding *BaseSimulationDataQuery* class which can be used to query for simulation data which matches a set of particular criteria within a storage backend, which in part includes querying for data collected:

- at a given `thermodynamic_state` (i.e temperature and pressure).
- for a given `property_phase` (e.g. gas, liquid, liquid+gas coexisting, …).
- using a given set of force field parameters identified by their unique `force_field_id` assigned by the storage system

Additionally included is not only the ability to find data generated for a particular `substance` (e.g. only data for methanol), but also the ability to return data for each component of a given substance by setting the *substance_query* attribute to a *SubstanceQuery* which has the *components_only* attribute set to true:

```
# Load an existing storage backend
storage_backend = LocalFileStorage()

# Define a system of 50% water and 50% methanol.
full_substance = Substance.from_components("O", "CO")

# Look for all simulation data generated for the full substance
data_query = SimulationDataQuery()

data_query.substance = full_substance
data_query.property_phase = PropertyPhase.Liquid

full_substance_data = storage_backend.query(data_query)

# Now look for all of the pure data which has been stored for both pure
# water and pure methanol.
pure_substance_query = SubstanceQuery()
pure_substance_query.components_only = True

data_query.substance_query = pure_substance_query
component_data = storage_backend.query(data_query)
```

This is particularly useful for when retrieving data for use in the calculation of excess properties (such as the enthalpy of mixing), where such calculations require information about both the full mixture as well as the pure components.

Single Simulation Data

The `StoredSimulationData` class is used to store data generated by a *single* molecular simulation and can be queried for using its accompanying `SimulationDataQuery` query class. In addition to the data stored by the parent `BaseSimulationData` class, this class further stores:

- the number of molecules which were simulated.
- the topology of the simulated system (stored as ancillary data).
- and trajectory of configurations (stored as ancillary data) and observables generated by the simulation.
- the statistic inefficiency of the data.

Data of this kind is considered replaceable, whereby data which has the lowest statistical efficiency is preferred. The philosophy here is that we should store the maximum amount of samples (i.e the maximum number of uncorrelated samples for the property which has the shortest correlation time) which will be useful for future calculations, such that future calaculations can simply discard the data which cannot be used (i.e. is likely correlated).

Free Energy Data

The `StoredFreeEnergyData` class is used to store data generated by a free energy calculation which computes the free energy difference between an end and start state. It can be queried for using its accompanying `FreeEnergyDataQuery` query class.

In addition to the data stored by the parent `BaseSimulationData` class, this class further stores:

- the free energy difference between the end and starting state.
- the topology of the system (stored as ancillary data).
- and trajectory of configurations generated in the starting and end states (stored as ancillary data).

Although data of this kind inherits from the `ReplaceableData` base class, all data deposited in a storage backend will be retained. At this time no situation can be envisaged that the same free energy data from exactly the same calculation will be stored, with the exception of operator errors.

2.30 Local File Storage

The `LocalFileStorage` backend stores and retrieves all data objects to / from the local file system. The root directory in which all data is to be stored is defined when the object is created:

```
storage_backend = LocalFileStorage(root_directory="stored_data")
```

All data objects will be stored within this directory as JSON files, with file names of the storage key assigned to that object. If the data object has an associated ancillary data directory, this will be **moved** (not copied) into the root directory and renamed to the storage key when that object is stored into the system.

An example directory created by a local storage backend will look something similar to:

```
- root_directory

    - 1fe615c5cb48429ab77fd71125dec297
        - trajectory.dcd
        - statistics.csv

    - 3e15d19e0e614d0491a1a0bc9a51534e
```

(continues on next page)

(continued from previous page)

```
        - trajectory.dcd
        - statistics.csv

    - 1fe615c5cb48429ab77fd71125dec297.json
    - 3e15d19e0e614d0491a1a0bc9a51534e.json
    - 0f71f2b4a22042d89d6f0882406869b6.json
```

where here the backend contains two data objects with ancillary data directories, and one without.

When retrieving data which has an ancillary data directory from the backend, the returned directory path will be the full path to the directory in the root storage directory.

2.31 Building the Docs

Although documentation for the OpenFF Evaluator is readily available online, it is sometimes useful to build a local version such as when

- developing new pages which you wish to preview without having to wait for ReadTheDocs to finish building.
- debugging errors which occur when building on ReadTheDocs.

In these cases, the docs can be built locally by doing the following:

```
git clone https://github.com/openforcefield/openff-evaluator.git
cd openff-evaluator/docs
conda env create --name openff-evaluator-docs --file environment.yaml
conda activate openff-evaluator-docs
rm -rf api && make clean && make html
```

The above will yield a new directory named *_build* which will contain the built html files which can be viewed in your local browser.

2.32 API

Documentation for each of the classes contained within the *openff.evaluator* framework.

2.32.1 Client Side API

`EvaluatorClient`	The object responsible for connecting to, and submitting physical property estimation requests to an *EvaluatorServer*.
`BatchMode`	The different modes in which a server can batch together properties to estimate.
`ConnectionOptions`	The options to use when connecting to an *EvaluatorServer*
`Request`	An estimation request which has been sent to a *EvaluatorServer* instance.
`RequestOptions`	The options to use when requesting a set of physical properties be estimated by the server.

continues on next page

Table 2 – continued from previous page

RequestResult	The current results of an estimation request - these results may be partial if the server hasn't yet completed the request.

EvaluatorClient

class openff.evaluator.client.**EvaluatorClient**(*connection_options=None*)

The object responsible for connecting to, and submitting physical property estimation requests to an *EvaluatorServer*.

Examples

These examples assume that an *EvaluatorServer* has been set up and is running (either synchronously or asynchronously). This server can be connect to be creating an *EvaluatorClient*:

```
>>> from openff.evaluator.client import EvaluatorClient
>>> property_estimator = EvaluatorClient()
```

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

```
>>> from openff.evaluator.client import ConnectionOptions
>>>
>>> connection_options = ConnectionOptions(server_address='server_address',
>>>                                        server_port=8000)
>>> property_estimator = EvaluatorClient(connection_options)
```

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

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from openff.evaluator.datasets.thermoml 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 openff.evaluator import unit
>>> from openff.evaluator.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 in the force field parameters
>>> from openff.evaluator.forcefield import SmirnoffForceFieldSource
>>> force_field_source = SmirnoffForceFieldSource.from_path('smirnoff99Frosst-1.1.0.
↪offxml')
>>>
>>> # Submit the estimation request to a running server.
>>> request = property_estimator.request_estimate(data_set, force_field_source)
```

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 *RequestOptions* object to the estimate commands.

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

```
>>> from openff.evaluator.layers.reweighting import ReweightingLayer
>>> from openff.evaluator.layers.simulation import SimulationLayer
>>>
>>> options = RequestOptions(calculation_layers=[
>>>     "ReweightingLayer",
>>>     "SimulationLayer"
>>> ])
>>>
>>> request = property_estimator.request_estimate(data_set, force_field_source,␣
↪options)
```

Options for how properties should be estimated can be set on a per property, and per layer basis by providing a calculation schema to the options object.

```
>>> from openff.evaluator.properties import DielectricConstant
>>>
>>> # Generate a schema to use when estimating densities directly
>>> # from simulations.
>>> density_simulation_schema = Density.default_simulation_schema()
>>> # Generate a schema to use when estimating dielectric constants
>>> # from cached simulation data.
>>> dielectric_reweighting_schema = DielectricConstant.default_reweighting_schema()
>>>
>>> options.workflow_options = {
>>>     'Density': {'SimulationLayer': density_simulation_schema},
>>>     'Dielectric': {'SimulationLayer': dielectric_reweighting_schema}
>>> }
>>>
>>> property_estimator.request_estimate(
>>>     data_set,
>>>     force_field_source,
>>>     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 openff.evaluator.forcefield import ParameterGradientKey
>>>
>>> parameter_gradient_keys = [
>>>     ParameterGradientKey('Bonds', '[#6:1]-[#8:2]', 'length')
```

(continues on next page)

(continued from previous page)

```
>>>     ParameterGradientKey('Angles', '[*:1]-[#8:2]-[*:3]', 'angle')
>>> ]
>>>
>>> property_estimator.request_estimate(
>>>     data_set,
>>>     force_field_source,
>>>     options,
>>>     parameter_gradient_keys
>>> )
```

__init__(*connection_options=None*)

Parameters **connection_options** (ConnectionOptions, *optional*) – The options used when connecting to the calculation server. If *None*, default options are used.

Methods

__init__([connection_options])	**param connection_options** The options used when connecting to the calculation
default_request_options(data_set, ...)	Returns the default *RequestOptions* options used to estimate a set of properties if *None* are provided.
request_estimate(property_set, ...[, ...])	Submits a request for the *EvaluatorServer* to attempt to estimate the data set of physical properties using the specified force field parameters according to the provided options.
retrieve_results(request_id[, synchronous, ...])	Retrieves the current results of a request from the server.

Attributes

server_address	The address of the server that this client is connected to.
server_port	The port of the server that this client is connected to.

property server_address
The address of the server that this client is connected to.

Type str

property server_port
The port of the server that this client is connected to.

Type int

static default_request_options(*data_set*, *force_field_source*)
Returns the default *RequestOptions* options used to estimate a set of properties if *None* are provided.

Parameters

- **data_set** (`PhysicalPropertyDataSet`) – The data set which would be estimated.
- **force_field_source** (`ForceFieldSource`) – The force field parameters which will be used by the request.

Returns The default options.

Return type *RequestOptions*

request_estimate(*property_set*, *force_field_source*, *options=None*, *parameter_gradient_keys=None*)
Submits a request for the *EvaluatorServer* to attempt to estimate the data set of physical properties using the specified force field parameters according to the provided options.

Parameters

- **property_set** (`PhysicalPropertyDataSet`) – The set of properties to estimate.
- **force_field_source** (`ForceFieldSource` *or openff.toolkit.typing.engines.smirnoff.ForceField*) – The force field parameters to estimate the properties using.
- **options** (`RequestOptions`, *optional*) – A set of estimator options. If *None* default options will be used (see *default_request_options*).
- **parameter_gradient_keys** (*list of ParameterGradientKey, optional*) – A list of the parameters that the physical properties should be differentiated with respect to.

Returns

- *Request* – An object which will provide access to the results of this request.
- *EvaluatorException, optional* – Any exceptions raised while attempting the submit the request.

retrieve_results(*request_id*, *synchronous=False*, *polling_interval=5*)
Retrieves the current results of a request from the server.

Parameters

- **request_id** (*str*) – The server assigned id of the request.
- **synchronous** (*bool*) – If true, this method will block the main thread until the server either returns a result or an error.
- **polling_interval** (*float*) – If running synchronously, this is the time interval (seconds) between checking if the request has completed.

Returns

- *RequestResult, optional* – Returns the current results of the request. This may be *None* if any unexpected exceptions occurred while retrieving the estimate.
- *EvaluatorException, optional* – The exception raised will trying to retrieve the result, if any.

BatchMode

class openff.evaluator.client.**BatchMode**(*value*)

The different modes in which a server can batch together properties to estimate.

This enum may take values of

- SameComponents: All properties measured for substances containing exactly the same components will be placed into a single batch. E.g. The density of a 80:20 and a 20:80 mix of ethanol and water would be batched together, but the density of pure ethanol and the density of pure water would be placed into separate batches.
- SharedComponents: All properties measured for substances containing at least common component will be batched together. E.g.The densities of 80:20 and 20:80 mixtures of ethanol and water, and the pure densities of ethanol and water would be batched together.

Properties will only be marked as estimated by the server when all properties in a single batch are completed.

__init__()

Attributes

SameComponents	
SharedComponents	

ConnectionOptions

class openff.evaluator.client.**ConnectionOptions**(*server_address=None*, *server_port=None*)

The options to use when connecting to an *EvaluatorServer*

__init__(*server_address=None*, *server_port=None*)

Parameters

- **server_address** (*str*) – The address of the server to connect to.
- **server_port** (*int*) – The port of the server to connect to.

Methods

__init__([server_address, server_port])	**param server_address** The address of the server to connect to.
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
validate([attribute_type])	Validate the values of the attributes.

Attributes

server_address	The address of the server to connect to.
server_port	The port of the server to connect to.

server_address
The address of the server to connect to. The default value of this attribute is `localhost`.

Type str

server_port
The port of the server to connect to. The default value of this attribute is `8000`.

Type int

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

Request

`class openff.evaluator.client.Request(client=None)`

An estimation request which has been sent to a *EvaluatorServer* instance.

This object can be used to query and retrieve the results of the request when finished, or be stored to retrieve the request at some point in the future.

`__init__(client=None)`

Parameters `client` (`EvaluatorClient`, *optional*) – The client which submitted this request.

Methods

`__init__`([client])	**param client** The client which submitted this request.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`results`([synchronous, polling_interval])	Attempt to retrieve the results of the request from the server.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`connection_options`	The options used to connect to the server handling the request.
`id`	The unique id assigned to this request by the server.

`id`

The unique id assigned to this request by the server. The default value of this attribute is not set and must be set by the user..

Type str

`connection_options`

The options used to connect to the server handling the request. The default value of this attribute is not set and must be set by the user..

Type *ConnectionOptions*

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

Attempt to retrieve the results of the request from the server.

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

Parameters

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

Returns

- *RequestResult, optional* – Returns the current results of the request. This may be *None* if any unexpected exceptions occurred while retrieving the estimate.
- *EvaluatorException, optional* – The exception raised will trying to retrieve the result if any.

`classmethod from_json(file_path)`
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes(attribute_type=None)`
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json(file_path=None, format=False)`
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json(string_contents)`
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`validate(attribute_type=None)`
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

RequestOptions

class openff.evaluator.client.**RequestOptions**

The options to use when requesting a set of physical properties be estimated by the server.

__init__()

Methods

`__init__`()	
`add_schema`(layer_type, property_type, schema)	A convenience function for adding a calculation schema to the schema dictionary.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`batch_mode`	The way in which the server should batch together properties to estimate.
`calculation_layers`	The calculation layers which may be used to estimate the set of physical properties.
`calculation_schemas`	The schemas that each calculation layer should use when estimating the set of physical properties.

calculation_layers

The calculation layers which may be used to estimate the set of physical properties. The order in which the layers appears in this list determines the order in which the layers will attempt to estimate the data set. The default value of this attribute is `['ReweightingLayer', 'SimulationLayer']`.

Type list

calculation_schemas

The schemas that each calculation layer should use when estimating the set of physical properties. The dictionary should be of the form [property_type][layer_type]. The default value of this attribute is not set. This attribute is *optional*.

Type dict

batch_mode

The way in which the server should batch together properties to estimate. Properties will only be marked as finished when all properties in a single batch are completed. The default value of this attribute is `BatchMode.SharedComponents`. This attribute is *optional*.

Type *BatchMode*

add_schema(*layer_type*, *property_type*, *schema*)

A convenience function for adding a calculation schema to the schema dictionary.

Parameters

- **layer_type** (*str or type of CalculationLayer*) – The layer to associate the schema with.
- **property_type** (*str or type of PhysicalProperty*) – The class of property to associate the schema with.
- **schema** (*CalculationSchema*) – The schema to add.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

classmethod **from_json**(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod **get_attributes**(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod **parse_json**(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

RequestResult

class openff.evaluator.client.**RequestResult**

The current results of an estimation request - these results may be partial if the server hasn't yet completed the request.

`__init__`()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`estimated_properties`	The set of properties which have been successfully estimated.
`exceptions`	The set of properties which have yet to be, or are currently being estimated.
`queued_properties`	The set of properties which have yet to be, or are currently being estimated.
`unsuccessful_properties`	The set of properties which could not be successfully estimated.

`queued_properties`

The set of properties which have yet to be, or are currently being estimated.

Type *PhysicalPropertyDataSet*

`estimated_properties`

The set of properties which have been successfully estimated.

Type *PhysicalPropertyDataSet*

`unsuccessful_properties`

The set of properties which could not be successfully estimated.

Type *PhysicalPropertyDataSet*

`exceptions`

The set of properties which have yet to be, or are currently being estimated. The default value of this attribute is `[]`.

Type list

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

Exceptions

EvaluatorException	A serializable wrapper around an *Exception*.

EvaluatorException

exception openff.evaluator.utils.exceptions.**EvaluatorException**(*message=None*)
A serializable wrapper around an *Exception*.

classmethod from_exception(*exception*)
Initialize this class from an existing exception.

Parameters **exception** (*Exception*) – The existing exception

Returns The initialized exception object.

Return type cls

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

2.32.2 Server Side API

EvaluatorServer	The object responsible for coordinating all properties estimations to be ran using the openff-evaluator framework.
Batch	Represents a batch of physical properties which are being estimated by the server for a given set of force field parameters.

EvaluatorServer

class openff.evaluator.server.**EvaluatorServer**(*calculation_backend*, *storage_backend=None*, *port=8000*, *working_directory='working-data'*, *enable_data_caching=True*, *delete_working_files=True*)

The object responsible for coordinating all properties estimations to be ran using the openff-evaluator framework.

This server is responsible for receiving estimation requests from the client, determining which calculation layer to use to launch the request, and distributing that estimation across the available compute resources.

Notes

Every client request is split into logical chunk batches. This enables batches of related properties (e.g. all properties for CO) to be estimated in one go (or one task graph in the case of workflow based layers) and returned when ready, rather than waiting for the full data set to complete.

Examples

Setting up a general server instance using a dask based calculation backend, and a local file storage backend:

```
>>> # Create the backend which will be responsible for distributing the calculations
>>> from openff.evaluator.backends.dask import DaskLocalCluster
>>> calculation_backend = DaskLocalCluster()
>>> calculation_backend.start()
>>>
>>> # Create the server to which all estimation requests will be submitted
>>> from openff.evaluator.server import EvaluatorServer
>>> property_server = EvaluatorServer(calculation_backend)
>>>
>>> # Instruct the server to listen for incoming requests
>>> # This command will run until killed.
>>> property_server.start()
```

`__init__`(*calculation_backend*, *storage_backend=None*, *port=8000*, *working_directory='working-data'*, *enable_data_caching=True*, *delete_working_files=True*)

Constructs a new EvaluatorServer object.

Parameters

- **`calculation_backend`** (`CalculationBackend`) – The backend to use for executing calculations.
- **`storage_backend`** (`StorageBackend, optional`) – The backend to use for storing information from any calculations. If *None*, a default *LocalFileStorage* backend will be used.
- **`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.
- **`enable_data_caching`** (*`bool`*) – Whether the server should attempt to cache any data, mainly the output of simulations, produced by estimation requests for future re-processing (e.g for reweighting).
- **`delete_working_files`** (*`bool`*) – Whether to delete the working files produced while estimated a batch of properties using a specific calculation layer.

Methods

`__init__`(calculation_backend[, ...])	Constructs a new EvaluatorServer object.
`start`([asynchronous])	Instructs the server to begin listening for incoming requests from any *EvaluatorClients*.
`stop`()	Stops the property calculation server and it's provided backend.

start(*asynchronous=False*)
Instructs the server to begin listening for incoming requests from any *EvaluatorClients*.

Parameters asynchronous (`bool`) – If *True* the server will run on a separate thread in the background, returning control back to the main thread. Otherwise, this function will block the main thread until this server is killed.

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

Batch

class openff.evaluator.server.**Batch**
Represents a batch of physical properties which are being estimated by the server for a given set of force field parameters.

The expectation is that this object will be passed between calculation layers, whereby each layer will attempt to estimate each of the *queued_properties*. Those properties which can be estimated will be moved to the *estimated_properties* set, while those that couldn't will remain in the *queued_properties* set ready for the next layer.

__init__()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

enable_data_caching	Whether the server should attempt to cache any data, mainly the output of simulations, produced by this batch for future re-processing (e.g for reweighting).
estimated_properties	The set of properties which have been successfully estimated.
exceptions	The set of properties which have yet to be, or are currently being estimated.
force_field_id	The id of the force field being used to estimatethis batch of properties.
id	The unique id of this batch.
options	The options being used to estimate this batch.
parameter_gradient_keys	The parameters that this batch of physical properties should be differentiated with respect to.
queued_properties	The set of properties which have yet to be estimated.
unsuccessful_properties	The set of properties which have been could not be estimated.

id

The unique id of this batch.

Type str

force_field_id

The id of the force field being used to estimatethis batch of properties. The default value of this attribute is not set and must be set by the user..

Type str

options

The options being used to estimate this batch. The default value of this attribute is not set and must be set by the user..

Type *RequestOptions*

parameter_gradient_keys

The parameters that this batch of physical properties should be differentiated with respect to. The default value of this attribute is not set and must be set by the user..

Type list

enable_data_caching

Whether the server should attempt to cache any data, mainly the output of simulations, produced by this batch for future re-processing (e.g for reweighting). The default value of this attribute is `True`.

Type bool

queued_properties

The set of properties which have yet to be estimated. The default value of this attribute is `[]`.

Type list

estimated_properties

The set of properties which have been successfully estimated. The default value of this attribute is `[]`.

Type list

`unsuccessful_properties`
The set of properties which have been could not be estimated. The default value of this attribute is `[]`.

Type list

`exceptions`
The set of properties which have yet to be, or are currently being estimated. The default value of this attribute is `[]`.

Type list

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters `attribute_type` (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters `file_path` (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters `attribute_type` (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- `file_path` (*str, optional*) – The (optional) file path to save the JSON file to.
- `format` (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters `string_contents` (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

2.32.3 Physical Property API

PhysicalProperty	Represents the value of any physical property and it's uncertainty if provided.
PropertyPhase	An enum describing the phase that a property was collected in.
Source	Container class for information about how a property was measured / calculated.
CalculationSource	Contains any metadata about how a physical property was calculated.
MeasurementSource	Contains any metadata about how a physical property was measured by experiment.

PhysicalProperty

class openff.evaluator.datasets.**PhysicalProperty**(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=None*, *source=None*)

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

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=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=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** (*openff.evaluator.unit.Quantity*) – The value of the measured physical property.
- **uncertainty** (*openff.evaluator.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.
default_unit()	openff.evaluator.unit.Unit: The default unit (e.g.
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
validate([attribute_type])	Validate the values of the attributes.

Attributes

gradients	The gradients of this property with respect to different force field parameters.
id	A unique identifier string assigned to this property
metadata	Additional metadata associated with this property.
phase	The phase / phases that this property was measured in.
source	The original source of this physical property.
substance	The substance that this property was measured estimated for.
thermodynamic_state	The thermodynamic state that this propertywas measured / estimated at.
uncertainty	The uncertainty in measured / estimated value of this property.
value	The measured / estimated value of this property.

abstract classmethod default_unit()
openff.evaluator.unit.Unit: The default unit (e.g. g / mol) associated with this class of property.

id
A unique identifier string assigned to this property

Type str

metadata
Additional metadata associated with this property. All property metadata will be made accessible to estimation workflows. The default value of this attribute is not set. This attribute is *optional*.

Type dict

thermodynamic_state
The thermodynamic state that this propertywas measured / estimated at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

phase
The phase / phases that this property was measured in. The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

substance
The substance that this property was measured estimated for. The default value of this attribute is not set and must be set by the user..

Type *Substance*

value
The measured / estimated value of this property. The default value of this attribute is not set and must be set by the user..

Type Quantity

uncertainty
The uncertainty in measured / estimated value of this property. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

`gradients`
The gradients of this property with respect to different force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type list

`source`
The original source of this physical property. The default value of this attribute is not set. This attribute is *optional*.

Type *Source*

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

PropertyPhase

class openff.evaluator.datasets.**PropertyPhase**(*value*)

An enum describing the phase that a property was collected in.

Examples

Properties measured in multiple phases (e.g. enthalpies of vaporization) can be defined be concatenating *PropertyPhase* enums:

```
>>> gas_liquid_phase = PropertyPhase.Gas | PropertyPhase.Liquid
```

__init__()

Methods

from_string(enum_string)	Parses a phase enum from its string representation.

Attributes

Undefined
Solid
Liquid
Gas

classmethod from_string(*enum_string*)

Parses a phase enum from its string representation.

Parameters **enum_string** (*str*) – The str representation of a *PropertyPhase*

Returns The created enum

Return type *PropertyPhase*

Examples

To round-trip convert a phase enum: >>> phase = PropertyPhase.Liquid | PropertyPhase.Gas >>> phase_str = str(phase) >>> parsed_phase = PropertyPhase.from_string(phase_str)

Source

class `openff.evaluator.datasets.`**`Source`**

Container class for information about how a property was measured / calculated.

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

`__init__`()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.

classmethod `from_json`(*file_path*)

Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod `parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

CalculationSource

class `openff.evaluator.datasets.`**`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.
`from_json`(file_path)	Create this object from a JSON file.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.

classmethod **`from_json`**(*file_path*)

Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod **`parse_json`**(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

MeasurementSource

class `openff.evaluator.datasets.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.
`from_json`(file_path)	Create this object from a JSON file.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.

classmethod **`from_json`**(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

Built-in Properties

Density	A class representation of a density property
ExcessMolarVolume	A class representation of an excess molar volume 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
SolvationFreeEnergy	A class representation of a solvation free energy property.
HostGuestBindingAffinity	A class representation of a host-guest binding affinity property

Density

class openff.evaluator.properties.**Density**(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=None*, *source=None*)

A class representation of a density property

__init__(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=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** (*openff.evaluator.unit.Quantity*) – The value of the measured physical property.
- **uncertainty** (*openff.evaluator.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.
`default_reweighting_schema`([...])	Returns the default calculation schema to use when estimating this property by reweighting existing data.
`default_simulation_schema`([...])	Returns the default calculation schema to use when estimating this class of property from direct simulations.
`default_unit`()	openff.evaluator.unit.Unit: The default unit (e.g.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`gradients`	The gradients of this property with respect to different force field parameters.
`id`	A unique identifier string assigned to this property
`metadata`	Additional metadata associated with this property.
`phase`	The phase / phases that this property was measured in.
`source`	The original source of this physical property.
`substance`	The substance that this property was measured estimated for.
`thermodynamic_state`	The thermodynamic state that this propertywas measured / estimated at.
`uncertainty`	The uncertainty in measured / estimated value of this property.
`value`	The measured / estimated value of this property.

classmethod default_unit()

openff.evaluator.unit.Unit: The default unit (e.g. g / mol) associated with this class of property.

static default_simulation_schema(*absolute_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *relative_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *n_molecules=1000*) → *openff.evaluator.layers.simulation.SimulationSchema*

Returns the default calculation schema to use when estimating this class of property from direct simulations.

Parameters

- **absolute_tolerance** (*openff.evaluator.unit.Quantity, optional*) – The absolute tolerance to estimate the property to within.
- **relative_tolerance** (*float*) – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.
- **n_molecules** (*int*) – The number of molecules to use in the simulation.

Returns The schema to follow when estimating this property.

Return type *SimulationSchema*

static default_reweighting_schema(*absolute_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>, relative_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>, n_effective_samples=50*) → *openff.evaluator.layers.reweighting.ReweightingSchema*

Returns the default calculation schema to use when estimating this property by reweighting existing data.

Parameters

- **absolute_tolerance** (*openff.evaluator.unit.Quantity, optional*) – The absolute tolerance to estimate the property to within.
- **relative_tolerance** (*float*) – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.
- **n_effective_samples** (*int*) – The minimum number of effective samples to require when reweighting the cached simulation data.

Returns The schema to follow when estimating this property.

Return type *ReweightingSchema*

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

gradients

The gradients of this property with respect to different force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type list

id

A unique identifier string assigned to this property

Type str

json(*file_path=None, format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`metadata`
Additional metadata associated with this property. All property metadata will be made accessible to estimation workflows. The default value of this attribute is not set. This attribute is *optional*.

Type dict

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`phase`
The phase / phases that this property was measured in. The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

`source`
The original source of this physical property. The default value of this attribute is not set. This attribute is *optional*.

Type *Source*

`substance`
The substance that this property was measured estimated for. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`thermodynamic_state`
The thermodynamic state that this propertywas measured / estimated at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

`uncertainty`
The uncertainty in measured / estimated value of this property. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

`value`
The measured / estimated value of this property. The default value of this attribute is not set and must be set by the user..

Type Quantity

ExcessMolarVolume

class openff.evaluator.properties.**ExcessMolarVolume**(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=None*, *source=None*)

A class representation of an excess molar volume property

__init__(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=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** (*openff.evaluator.unit.Quantity*) – The value of the measured physical property.
- **uncertainty** (*openff.evaluator.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.
default_reweighting_schema([...])	Returns the default calculation schema to use when estimating this class of property by re-weighting cached simulation data.
default_simulation_schema([...])	Returns the default calculation schema to use when estimating this class of property from direct simulations.
default_unit()	openff.evaluator.unit.Unit: The default unit (e.g.
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
validate([attribute_type])	Validate the values of the attributes.

Attributes

gradients	The gradients of this property with respect to different force field parameters.
id	A unique identifier string assigned to this property
metadata	Additional metadata associated with this property.
phase	The phase / phases that this property was measured in.
source	The original source of this physical property.
substance	The substance that this property was measured estimated for.
thermodynamic_state	The thermodynamic state that this propertywas measured / estimated at.
uncertainty	The uncertainty in measured / estimated value of this property.
value	The measured / estimated value of this property.

classmethod default_unit()

openff.evaluator.unit.Unit: The default unit (e.g. g / mol) associated with this class of property.

classmethod default_reweighting_schema(*absolute_tolerance: openff.evaluator.utils.units.Quantity = <openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *relative_tolerance: float = <openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *n_effective_samples: int = 50*) → *openff.evaluator.layers.reweighting.ReweightingSchema*

Returns the default calculation schema to use when estimating this class of property by re-weighting cached simulation data.

Parameters

- **absolute_tolerance** – The absolute tolerance to estimate the property to within.
- **relative_tolerance** – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.
- **n_effective_samples** – The minimum number of effective samples to require when reweighting the cached simulation data.

Returns

Return type The default re-weighting calculation schema.

classmethod default_simulation_schema(*absolute_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *relative_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *n_molecules=1000*) → *openff.evaluator.layers.simulation.SimulationSchema*

Returns the default calculation schema to use when estimating this class of property from direct simulations.

Parameters

- **absolute_tolerance** (*openff.evaluator.unit.Quantity, optional*) – The absolute tolerance to estimate the property to within.
- **relative_tolerance** (*float*) – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.

- **n_molecules** (*int*) – The number of molecules to use in the simulation.

Returns The schema to follow when estimating this property.

Return type *SimulationSchema*

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

gradients
The gradients of this property with respect to different force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type list

id
A unique identifier string assigned to this property

Type str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

metadata
Additional metadata associated with this property. All property metadata will be made accessible to estimation workflows. The default value of this attribute is not set. This attribute is *optional*.

Type dict

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

phase
The phase / phases that this property was measured in. The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

`source`
The original source of this physical property. The default value of this attribute is not set. This attribute is *optional*.

Type *Source*

`substance`
The substance that this property was measured estimated for. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`thermodynamic_state`
The thermodynamic state that this propertywas measured / estimated at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

`uncertainty`
The uncertainty in measured / estimated value of this property. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **`or`** `AssertionError` –

`value`
The measured / estimated value of this property. The default value of this attribute is not set and must be set by the user..

Type Quantity

DielectricConstant

`class openff.evaluator.properties.`**`DielectricConstant`**(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=None*, *source=None*)
A class representation of a dielectric property

`__init__`(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=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`** (*`openff.evaluator.unit.Quantity`*) – The value of the measured physical property.

- **uncertainty** (*openff.evaluator.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.
default_reweighting_schema([...])	Returns the default calculation schema to use when estimating this property by reweighting existing data.
default_simulation_schema([...])	Returns the default calculation schema to use when estimating this class of property from direct simulations.
default_unit()	openff.evaluator.unit.Unit: The default unit (e.g.
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
validate([attribute_type])	Validate the values of the attributes.

Attributes

gradients	The gradients of this property with respect to different force field parameters.
id	A unique identifier string assigned to this property
metadata	Additional metadata associated with this property.
phase	The phase / phases that this property was measured in.
source	The original source of this physical property.
substance	The substance that this property was measured estimated for.
thermodynamic_state	The thermodynamic state that this propertywas measured / estimated at.
uncertainty	The uncertainty in measured / estimated value of this property.
value	The measured / estimated value of this property.

classmethod default_unit()

openff.evaluator.unit.Unit: The default unit (e.g. g / mol) associated with this class of property.

static default_simulation_schema(*absolute_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>, relative_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>, n_molecules=1000*)

Returns the default calculation schema to use when estimating this class of property from direct simulations.

Parameters

- **absolute_tolerance** (*openff.evaluator.unit.Quantity, optional*) – The absolute tolerance to estimate the property to within.

- **relative_tolerance** (*float*) – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.
- **n_molecules** (*int*) – The number of molecules to use in the simulation.

Returns The schema to follow when estimating this property.

Return type *SimulationSchema*

static default_reweighting_schema(*absolute_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>, relative_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>, n_effective_samples=50*)

Returns the default calculation schema to use when estimating this property by reweighting existing data.

Parameters

- **absolute_tolerance** (*openff.evaluator.unit.Quantity, optional*) – The absolute tolerance to estimate the property to within.
- **relative_tolerance** (*float*) – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.
- **n_effective_samples** (*int*) – The minimum number of effective samples to require when reweighting the cached simulation data.

Returns The schema to follow when estimating this property.

Return type *ReweightingSchema*

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters file_path (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

gradients

The gradients of this property with respect to different force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type list

id

A unique identifier string assigned to this property

Type str

json(*file_path=None, format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.

- `format` (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`metadata`
Additional metadata associated with this property. All property metadata will be made accessible to estimation workflows. The default value of this attribute is not set. This attribute is *optional*.

Type dict

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters `string_contents` (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`phase`
The phase / phases that this property was measured in. The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

`source`
The original source of this physical property. The default value of this attribute is not set. This attribute is *optional*.

Type *Source*

`substance`
The substance that this property was measured estimated for. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`thermodynamic_state`
The thermodynamic state that this propertywas measured / estimated at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

`uncertainty`
The uncertainty in measured / estimated value of this property. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters `attribute_type` (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`value`
The measured / estimated value of this property. The default value of this attribute is not set and must be set by the user..

Type Quantity

EnthalpyOfMixing

class `openff.evaluator.properties.`**`EnthalpyOfMixing`**(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=None*, *source=None*)

A class representation of an enthalpy of mixing property

`__init__`(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=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`** (*`openff.evaluator.unit.Quantity`*) – The value of the measured physical property.
- **`uncertainty`** (*`openff.evaluator.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.
`default_reweighting_schema`([...])	Returns the default calculation schema to use when estimating this class of property by re-weighting cached simulation data.
`default_simulation_schema`([...])	Returns the default calculation schema to use when estimating this class of property from direct simulations.
`default_unit`()	openff.evaluator.unit.Unit: The default unit (e.g.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`gradients`	The gradients of this property with respect to different force field parameters.
`id`	A unique identifier string assigned to this property
`metadata`	Additional metadata associated with this property.
`phase`	The phase / phases that this property was measured in.
`source`	The original source of this physical property.
`substance`	The substance that this property was measured estimated for.
`thermodynamic_state`	The thermodynamic state that this propertywas measured / estimated at.
`uncertainty`	The uncertainty in measured / estimated value of this property.
`value`	The measured / estimated value of this property.

`classmethod default_unit()`

openff.evaluator.unit.Unit: The default unit (e.g. g / mol) associated with this class of property.

`classmethod default_reweighting_schema`(*absolute_tolerance: openff.evaluator.utils.units.Quantity = <openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *relative_tolerance: float = <openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *n_effective_samples: int = 50*) → *openff.evaluator.layers.reweighting.ReweightingSchema*

Returns the default calculation schema to use when estimating this class of property by re-weighting cached simulation data.

Parameters

- **`absolute_tolerance`** – The absolute tolerance to estimate the property to within.
- **`relative_tolerance`** – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.
- **`n_effective_samples`** – The minimum number of effective samples to require when reweighting the cached simulation data.

Returns

Return type The default re-weighting calculation schema.

`classmethod default_simulation_schema`(*absolute_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *relative_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *n_molecules=1000*) → *openff.evaluator.layers.simulation.SimulationSchema*

Returns the default calculation schema to use when estimating this class of property from direct simulations.

Parameters

- **`absolute_tolerance`** (*`openff.evaluator.unit.Quantity, optional`*) – The absolute tolerance to estimate the property to within.
- **`relative_tolerance`** (*`float`*) – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.

- **n_molecules** (*int*) – The number of molecules to use in the simulation.

Returns The schema to follow when estimating this property.

Return type *SimulationSchema*

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

gradients
The gradients of this property with respect to different force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type list

id
A unique identifier string assigned to this property

Type str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

metadata
Additional metadata associated with this property. All property metadata will be made accessible to estimation workflows. The default value of this attribute is not set. This attribute is *optional*.

Type dict

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

phase
The phase / phases that this property was measured in. The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

`source`
The original source of this physical property. The default value of this attribute is not set. This attribute is *optional*.

Type *Source*

`substance`
The substance that this property was measured estimated for. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`thermodynamic_state`
The thermodynamic state that this propertywas measured / estimated at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

`uncertainty`
The uncertainty in measured / estimated value of this property. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`value`
The measured / estimated value of this property. The default value of this attribute is not set and must be set by the user..

Type Quantity

EnthalpyOfVaporization

class `openff.evaluator.properties.`**`EnthalpyOfVaporization`**(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=None*, *source=None*)
A class representation of an enthalpy of vaporization property

`__init__`(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=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`** (*`openff.evaluator.unit.Quantity`*) – The value of the measured physical property.

- **uncertainty** (*openff.evaluator.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.
default_reweighting_schema([...])	Returns the default calculation schema to use when estimating this property by reweighting existing data.
default_simulation_schema([...])	Returns the default calculation schema to use when estimating this class of property from direct simulations.
default_unit()	openff.evaluator.unit.Unit: The default unit (e.g.
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
validate([attribute_type])	Validate the values of the attributes.

Attributes

gradients	The gradients of this property with respect to different force field parameters.
id	A unique identifier string assigned to this property
metadata	Additional metadata associated with this property.
phase	The phase / phases that this property was measured in.
source	The original source of this physical property.
substance	The substance that this property was measured estimated for.
thermodynamic_state	The thermodynamic state that this propertywas measured / estimated at.
uncertainty	The uncertainty in measured / estimated value of this property.
value	The measured / estimated value of this property.

classmethod default_unit()

openff.evaluator.unit.Unit: The default unit (e.g. g / mol) associated with this class of property.

static default_simulation_schema(*absolute_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *relative_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *n_molecules=1000*)

Returns the default calculation schema to use when estimating this class of property from direct simulations.

Parameters

- **absolute_tolerance** (*openff.evaluator.unit.Quantity, optional*) – The absolute tolerance to estimate the property to within.

- **relative_tolerance** (*float*) – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.
- **n_molecules** (*int*) – The number of molecules to use in the simulation.

Returns The schema to follow when estimating this property.

Return type *SimulationSchema*

classmethod default_reweighting_schema(*absolute_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *relative_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *n_effective_samples=50*)

Returns the default calculation schema to use when estimating this property by reweighting existing data.

Parameters

- **absolute_tolerance** (*openff.evaluator.unit.Quantity, optional*) – The absolute tolerance to estimate the property to within.
- **relative_tolerance** (*float*) – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.
- **n_effective_samples** (*int*) – The minimum number of effective samples to require when reweighting the cached simulation data.

Returns The schema to follow when estimating this property.

Return type *ReweightingSchema*

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

gradients

The gradients of this property with respect to different force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type list

id

A unique identifier string assigned to this property

Type str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.

- `format` (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

metadata
Additional metadata associated with this property. All property metadata will be made accessible to estimation workflows. The default value of this attribute is not set. This attribute is *optional*.

Type dict

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

phase
The phase / phases that this property was measured in. The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

source
The original source of this physical property. The default value of this attribute is not set. This attribute is *optional*.

Type *Source*

substance
The substance that this property was measured estimated for. The default value of this attribute is not set and must be set by the user..

Type *Substance*

thermodynamic_state
The thermodynamic state that this propertywas measured / estimated at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

uncertainty
The uncertainty in measured / estimated value of this property. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

value
The measured / estimated value of this property. The default value of this attribute is not set and must be set by the user..

Type Quantity

SolvationFreeEnergy

class openff.evaluator.properties.**SolvationFreeEnergy**(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=None*, *source=None*)

A class representation of a solvation free energy property.

__init__(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=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** (*openff.evaluator.unit.Quantity*) – The value of the measured physical property.
- **uncertainty** (*openff.evaluator.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.
default_simulation_schema([...])	Returns the default calculation schema to use when estimating this class of property from direct simulations.
default_unit()	openff.evaluator.unit.Unit: The default unit (e.g.
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
validate([attribute_type])	Validate the values of the attributes.

Attributes

gradients	The gradients of this property with respect to different force field parameters.
id	A unique identifier string assigned to this property
metadata	Additional metadata associated with this property.
phase	The phase / phases that this property was measured in.
source	The original source of this physical property.

continues on next page

Table 39 – continued from previous page

substance	The substance that this property was measured estimated for.
thermodynamic_state	The thermodynamic state that this propertywas measured / estimated at.
uncertainty	The uncertainty in measured / estimated value of this property.
value	The measured / estimated value of this property.

classmethod default_unit()
openff.evaluator.unit.Unit: The default unit (e.g. g / mol) associated with this class of property.

static default_simulation_schema(*absolute_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *relative_tolerance=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *n_molecules=2000*)
Returns the default calculation schema to use when estimating this class of property from direct simulations.

Parameters

- **absolute_tolerance** (*openff.evaluator.unit.Quantity, optional*) – The absolute tolerance to estimate the property to within.
- **relative_tolerance** (*float*) – The tolerance (as a fraction of the properties reported uncertainty) to estimate the property to within.
- **n_molecules** (*int*) – The number of molecules to use in the simulation.

Returns The schema to follow when estimating this property.

Return type *SimulationSchema*

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

gradients
The gradients of this property with respect to different force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type list

id
A unique identifier string assigned to this property

Type str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

metadata

Additional metadata associated with this property. All property metadata will be made accessible to estimation workflows. The default value of this attribute is not set. This attribute is *optional*.

Type dict

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters string_contents (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

phase

The phase / phases that this property was measured in. The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

source

The original source of this physical property. The default value of this attribute is not set. This attribute is *optional*.

Type *Source*

substance

The substance that this property was measured estimated for. The default value of this attribute is not set and must be set by the user..

Type *Substance*

thermodynamic_state

The thermodynamic state that this propertywas measured / estimated at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

uncertainty

The uncertainty in measured / estimated value of this property. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

value

The measured / estimated value of this property. The default value of this attribute is not set and must be set by the user..

Type Quantity

HostGuestBindingAffinity

class `openff.evaluator.properties.`**`HostGuestBindingAffinity`**(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=None*, *source=None*)

A class representation of a host-guest binding affinity property

`__init__`(*thermodynamic_state=None*, *phase=PropertyPhase.Undefined*, *substance=None*, *value=None*, *uncertainty=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`** (*`openff.evaluator.unit.Quantity`*) – The value of the measured physical property.
- **`uncertainty`** (*`openff.evaluator.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.
`default_paprika_schema`([existing_schema, ...])	Returns the default calculation schema to use when estimating a host-guest binding affinity measurement with an APR calculation using the `paprika` package.
`default_unit`()	openff.evaluator.unit.Unit: The default unit (e.g.
`default_yank_schema`([existing_schema])	Returns the default calculation schema to use when estimating this class of property from direct simulations.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

gradients	The gradients of this property with respect to different force field parameters.
id	A unique identifier string assigned to this property
metadata	Additional metadata associated with this property.
phase	The phase / phases that this property was measured in.
source	The original source of this physical property.
substance	The substance that this property was measured estimated for.
thermodynamic_state	The thermodynamic state that this propertywas measured / estimated at.
uncertainty	The uncertainty in measured / estimated value of this property.
value	The measured / estimated value of this property.

classmethod default_unit()

openff.evaluator.unit.Unit: The default unit (e.g. g / mol) associated with this class of property.

static default_yank_schema(*existing_schema=None*)

Returns the default calculation schema to use when estimating this class of property from direct simulations.

Parameters existing_schema (`SimulationSchema`, *optional*) – An existing schema whose settings to use. If set, the schema's *workflow_schema* will be overwritten by this method.

Returns The schema to follow when estimating this property.

Return type *SimulationSchema*

classmethod default_paprika_schema(*existing_schema: Optional[openff.evaluator.layers.simulation.SimulationSchema] = None*, *n_solvent_molecules: int = 2500*, *n_thermalization_steps: int = 50000*, *n_equilibration_steps: int = 200000*, *n_production_steps: int = 2500000*, *dt_thermalization: openff.evaluator.utils.units.Quantity = <Quantity(1.0, 'femtosecond')>*, *dt_equilibration: openff.evaluator.utils.units.Quantity = <Quantity(2.0, 'femtosecond')>*, *dt_production: openff.evaluator.utils.units.Quantity = <Quantity(2.0, 'femtosecond')>*, *debug: bool = False*)

Returns the default calculation schema to use when estimating a host-guest binding affinity measurement with an APR calculation using the `paprika` package.

Notes

- This schema requires additional metadata to be able to estimate each metadata. This metadata is automatically generated for properties loaded from the `taproom` package using the `TaproomDataSet` object.

  Parameters

  - **`existing_schema`** (`SimulationSchema`, *`optional`*) – An existing schema whose settings to use. If set, the schema's *workflow_schema* will be overwritten by this method.
  - **`n_solvent_molecules`** – The number of solvent molecules to add to the box.
  - **`n_thermalization_steps`** – The number of thermalization simulations steps to perform. Sample generated during this step will be discarded.
  - **`n_equilibration_steps`** – The number of equilibration simulations steps to perform. Sample generated during this step will be discarded.
  - **`n_production_steps`** – The number of production simulations steps to perform. Sample generated during this step will be used in the final free energy calculation.
  - **`dt_thermalization`** – The integration timestep during thermalization
  - **`dt_equilibration`** – The integration timestep during equilibration
  - **`dt_production`** – The integration timestep during production
  - **`debug`** – Whether to return a debug schema. This is nearly identical to the default schema, albeit with significantly less solvent molecules (10), all simulations run in NVT and much shorter simulation runs (500 steps). If True, the other input arguments will be ignored.

  Returns The schema to follow when estimating this property.

  Return type *SimulationSchema*

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`gradients`
The gradients of this property with respect to different force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type list

`id`
A unique identifier string assigned to this property

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`metadata`
Additional metadata associated with this property. All property metadata will be made accessible to estimation workflows. The default value of this attribute is not set. This attribute is *optional*.

Type dict

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`phase`
The phase / phases that this property was measured in. The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

`source`
The original source of this physical property. The default value of this attribute is not set. This attribute is *optional*.

Type *Source*

`substance`
The substance that this property was measured estimated for. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`thermodynamic_state`
The thermodynamic state that this propertywas measured / estimated at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

`uncertainty`
The uncertainty in measured / estimated value of this property. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

value
The measured / estimated value of this property. The default value of this attribute is not set and must be set by the user..

Type Quantity

Substance Definition

`Substance`	Defines the components, their amounts, and their roles in a system.
`Component`	Defines a single component in a chemical system, as well as it's role within the system (if any).
`Amount`	A representation of the amount of a given component in a *Substance*.
`ExactAmount`	The exact number of instances of a *Component* in a *Substance*.
`MoleFraction`	The mole fraction of a *Component* in a *Substance*.

Substance

class `openff.evaluator.substances.`**`Substance`**
Defines the components, their amounts, and their roles in a system.

Examples

A neat liquid containing only a single component:

```
>>> from openff.evaluator.substances import Component, ExactAmount, MoleFraction
>>> liquid = Substance()
>>> liquid.add_component(Component(smiles='O'), MoleFraction(1.0))
```

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

```
>>> binary_mixture = Substance()
>>> binary_mixture.add_component(Component(smiles='O'), MoleFraction(0.2))
>>> binary_mixture.add_component(Component(smiles='CO'), 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 = Component(smiles='C1=CC=CC=C1', role=Component.Role.Solute)
>>> water = Component(smiles='O', role=Component.Role.Solvent)
>>>
>>> infinite_dilution = Substance()
>>> infinite_dilution.add_component(component=benzene, amount=ExactAmount(1)) #␣
↪Infinite dilution.
>>> infinite_dilution.add_component(component=water, amount=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__`()

Methods

`__init__`()	
`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
`from_components`(*components)	Creates a new *Substance* object from a list of components.
`from_json`(file_path)	Create this object from a JSON file.
`get_amounts`(component)	Returns the amounts of the component in this substance.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`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`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`amounts`	the amounts of the component in this substance This attribute is *read-only*.
`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.

components

A list of all of the components in this substance. The default value of this attribute is `()`. This attribute is *read-only*.

Type tuple

amounts

the amounts of the component in this substance This attribute is *read-only*.

Type dict

property identifier

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

Type str

property number_of_components

The number of different components in this substance.

Type int

classmethod from_components(**components*)

Creates a new *Substance* object from a list of components. This method assumes that all components should be present with equal mole fractions.

Parameters **components** (`Component` *or* `str`) – The components to add to the substance. These may either be full *Component* objects or just the smiles representation of the component.

Returns The substance containing the requested components in equal amounts.

Return type *Substance*

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

get_amounts(*component*)

Returns the amounts of the component in this substance.

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

Returns The amounts of the component in this substance.

Return type tuple of Amount

get_molecules_per_component(*maximum_molecules*, *tolerance=None*, *count_exact_amount=True*, *truncate_n_molecules=True*)

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.
- **tolerance** (`float, optional`) – The tolerance within which this amount should be represented. As an example, when converting a mole fraction into a number of molecules, the total number of molecules may not be sufficiently large enough to reproduce this amount.
- **count_exact_amount** (`bool`) – Whether components present in an exact amount (i.e. defined with an `ExactAmount`) should be considered when apply the maximum number

  of molecules constraint. This may be set false, for example, when building a separate solvated protein (n = 1) and solvated protein + ligand complex (n = 2) system but wish for both systems to have the same number of solvent molecules.
- **truncate_n_molecules** (`bool`) – Whether or not to attempt to truncate the number of molecules in the substance if the total number is over the specified maximum. If False, an exception will be raised in this case.

  The truncation works by iteratively removing one molecule of the predominant component up to a limit of removing a total number of molecules equal to the number of components in the substance (e.g. for a binary substance a maximum of two molecules can be removed). An exception is raised if the number of molecules cannot be sensibly truncated.

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`** (*`openff.evaluator.unit.Quantity`*) – The ionic string in units of molar.

Returns The mole fraction of ions.

Return type float

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

Component

`class openff.evaluator.substances.Component`(*smiles=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *role=Role.Solvent*)

Defines a single component in a chemical system, as well as it's role within the system (if any).

`__init__`(*smiles=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *role=Role.Solvent*)

Constructs a new Component object with either a label or a smiles string, but not both.

Notes

The *label* and *smiles* arguments are mutually exclusive, and only one can be passed while the other should be *None*.

Parameters

- **smiles** (*str*) – A SMILES descriptor of the component
- **role** (`Component.Role`) – The role of this component in the system.

Methods

__init__([smiles, role])	Constructs a new Component object with either a label or a smiles string, but not both.
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
validate([attribute_type])	Validate the values of the attributes.

Attributes

identifier	A unique identifier for this component.
role	The role of this component in the system.
smiles	The SMILES pattern which describes this component.

`class Role`(*value*)

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 used by workflow to identify the correct species in a system, such as when doing docking or performing solvation free energy calculations.

`smiles`

The SMILES pattern which describes this component. The default value of this attribute is not set and must be set by the user.. This attribute is *read-only*.

Type str

`role`

The role of this component in the system. The default value of this attribute is `Role.Solvent`. This attribute is *read-only*.

Type *Component.Role*

property identifier
A unique identifier for this component.

Type str

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

Amount

class `openff.evaluator.substances.`**`Amount`**(*value=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*)

A representation of the amount of a given component in a *Substance*.

`__init__`(*value=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*)

Parameters **`value`** (*`float` or `int`*) – The value of this amount.

Methods

`__init__`([value])	**param value** The value of this amount.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`to_number_of_molecules`(total_substance_molecules)	Converts this amount to an exact number of molecules
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`identifier`	A string identifier for this amount.
`value`	The value of this amount.

`value`

The value of this amount. The default value of this attribute is not set and must be set by the user.. This attribute is *read-only*.

Type typing.Union[float, int]

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`, `optional`*) – 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 sufficiently large enough to reproduce this amount.

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

Return type int

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None, format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

ExactAmount

class openff.evaluator.substances.**ExactAmount**(*value=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*)
The exact number of instances of a *Component* in a *Substance*.

An assumption is made that this amount is for a component which is infinitely dilute (such as ligands in binding calculations), and hence do not contribute to the total mole fraction of a *Substance*.

__init__(*value=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*)

Parameters **value** (*float or int*) – The value of this amount.

Methods

`__init__`([value])	**param value** The value of this amount.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`to_number_of_molecules`(total_substance_molecules)	Converts this amount to an exact number of molecules
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`identifier`	A string identifier for this amount.
`value`	The value of this amount.

value
The value of this amount. The default value of this attribute is not set and must be set by the user..

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, optional*) – 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 sufficiently large enough to reproduce this amount.

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

Return type int

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters file_path (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (`str, optional`) – The (optional) file path to save the JSON file to.
- **`format`** (`bool`) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (`str or bytes`) – The typed json string.

Returns The parsed class.

Return type Any

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (`type of Attribute, optional`) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

MoleFraction

`class openff.evaluator.substances.MoleFraction`(*value=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*)

The mole fraction of a *Component* in a *Substance*.

`__init__`(*value=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*)

Parameters **`value`** (`float or int`) – The value of this amount.

Methods

`__init__`([value])	**param value** The value of this amount.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`to_number_of_molecules`(total_substance_molecules)	Converts this amount to an exact number of molecules
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

identifier	A string identifier for this amount.
value	The value of this amount.

value
The value of this amount. The default value of this attribute is not set and must be set by the user..

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, optional*) – 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 sufficiently large enough to reproduce this amount.

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

Return type int

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError or AssertionError –

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters file_path (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.

- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

State Definition

ThermodynamicState	Data specifying a physical thermodynamic state obeying Boltzmann statistics.

ThermodynamicState

class openff.evaluator.thermodynamics.**ThermodynamicState**(*temperature=None*, *pressure=None*)
Data specifying a physical thermodynamic state obeying Boltzmann statistics.

Notes

Equality of two thermodynamic states is determined by comparing the temperature in kelvin to within 3 decimal places, and comparing the pressure (if defined) in pascals to within 3 decimal places.

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** (*openff.evaluator.unit.Quantity*) – The external temperature
- **pressure** (*openff.evaluator.unit.Quantity*) – The external pressure

Methods

`__init__`([temperature, pressure])	Constructs a new ThermodynamicState object.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

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
`pressure`	The external pressure.
`temperature`	The external temperature.

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

temperature
The external temperature. The default value of this attribute is not set and must be set by the user..

Type Quantity

pressure
The external pressure. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod **parse_json**(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

2.32.4 Data Set API

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

PhysicalPropertyDataSet

class openff.evaluator.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.
add_properties(*physical_properties[, validate])	Adds a physical property to the data set.
from_json(file_path)	Create this object from a JSON file.
from_pandas(data_frame)	Constructs a data set object from a pandas `DataFrame` object.
json([file_path, format])	Creates a JSON representation of this class.
merge(data_set[, validate])	Merge another data set into the current one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
properties_by_substance(substance)	A generator which may be used to loop over all of the properties which were measured for a particular substance.

continues on next page

Table 57 – continued from previous page

`properties_by_type`(property_type)	A generator which may be used to loop over all of properties of a particular type, e.g.
`to_pandas`()	Converts a *PhysicalPropertyDataSet* to a *pandas.DataFrame* object with columns of
`validate`()	Checks to ensure that all properties within the set are valid physical property object.

Attributes

`properties`	A list of all of the properties within this set.
`property_types`	The types of property within this data set.
`sources`	The sources from which the properties in this data set were gathered.
`substances`	The substances for which the properties in this data set were collected for.

property properties
A list of all of the properties within this set.

Type tuple of PhysicalProperty

property property_types
The types of property within this data set.

Type set of str

property substances
The substances for which the properties in this data set were collected for.

Type set of Substance

property sources
The sources from which the properties in this data set were gathered.

Type set of Source

merge(*data_set*, *validate=True*)
Merge another data set into the current one.

Parameters

- **data_set** (`PhysicalPropertyDataSet`) – The secondary data set to merge into this one.
- **validate** (*bool*) – Whether to validate the other data set before merging.

add_properties(**physical_properties*, *validate=True*)
Adds a physical property to the data set.

Parameters

- **physical_properties** (`PhysicalProperty`) – The physical property to add.
- **validate** (*bool*) – Whether to validate the properties before adding them to the set.

properties_by_substance(*substance*)
A generator which may be used to loop over all of the properties which were measured for a particular substance.

Parameters **substance** (`Substance`) – The substance of interest.

Returns

Return type generator of PhysicalProperty

`properties_by_type`(*property_type*)

A generator which may be used to loop over all of properties of a particular type, e.g. all "Density" properties.

Parameters `property_type` (*str or type of PhysicalProperty*) – The type of property of interest. This may either be the string class name of the property or the class type.

Returns

Return type generator of PhysicalProperty

`validate`()

Checks to ensure that all properties within the set are valid physical property object.

`to_pandas`()

Converts a *PhysicalPropertyDataSet* to a *pandas.DataFrame* object with columns of

- 'Id'
- 'Temperature (K)'
- 'Pressure (kPa)'
- 'Phase'
- 'N Components'
- 'Component 1'
- 'Role 1'
- 'Mole Fraction 1'
- 'Exact Amount 1'
- …
- 'Component N'
- 'Role N'
- 'Mole Fraction N'
- 'Exact Amount N'
- '<Property 1> Value (<default unit>)'
- '<Property 1> Uncertainty / (<default unit>)'
- …
- '<Property N> Value / (<default unit>)'
- '<Property N> Uncertainty / (<default unit>)'
- *'Source'*

where 'Component X' is a column containing the smiles representation of component X.

Returns The create data frame.

Return type pandas.DataFrame

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_pandas(*data_frame: pandas.core.frame.DataFrame*) → *openff.evaluator.datasets.datasets.PhysicalPropertyDataSet*

Constructs a data set object from a pandas `DataFrame` object.

Notes

- All physical properties are assumed to be source from experimental measurements.
- Currently this method onlu supports data frames containing properties which are built-in to the framework (e.g. Density).
- This method assumes the data frame has a structure identical to that produced by the `PhysicalPropertyDataSet.to_pandas` function.

Parameters **data_frame** – The data frame to construct the data set from.

Returns

Return type The constructed data set.

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

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 function used to map a property from the ThermoML archive to an internal *PhysicalProperty* object of the correct type.
thermoml_property	A decorator which wraps around the *register_thermoml_property* method.

ThermoMLDataSet

`class openff.evaluator.datasets.thermoml.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.
`add_properties`(*physical_properties[, validate])	Adds a physical property to the data set.
`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_json`(file_path)	Create this object from a JSON file.
`from_pandas`(data_frame)	Constructs a data set object from a pandas `DataFrame` object.
`from_url`(*url_list)	Load a ThermoML data set from a list of URLs
`from_xml`(xml, default_source)	Load a ThermoML data set from an xml object.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(data_set[, validate])	Merge another data set into the current one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`properties_by_substance`(substance)	A generator which may be used to loop over all of the properties which were measured for a particular substance.
`properties_by_type`(property_type)	A generator which may be used to loop over all of properties of a particular type, e.g.
`to_pandas`()	Converts a *PhysicalPropertyDataSet* to a *pandas.DataFrame* object with columns of
`validate`()	Checks to ensure that all properties within the set are valid physical property object.

Attributes

properties	A list of all of the properties within this set.
property_types	The types of property within this data set.
registered_properties	
sources	The sources from which the properties in this data set were gathered.
substances	The substances for which the properties in this data set were collected for.

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*

add_properties(**physical_properties*, *validate=True*)
Adds a physical property to the data set.

Parameters

- **physical_properties** (PhysicalProperty) – The physical property to add.
- **validate** (*bool*) – Whether to validate the properties before adding them to the set.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_pandas(*data_frame: pandas.core.frame.DataFrame*) → *openff.evaluator.datasets.datasets.PhysicalPropertyDataSet*
Constructs a data set object from a pandas `DataFrame` object.

Notes

- All physical properties are assumed to be source from experimental measurements.
- Currently this method onlu supports data frames containing properties which are built-in to the framework (e.g. Density).
- This method assumes the data frame has a structure identical to that produced by the `PhysicalPropertyDataSet.to_pandas` function.

Parameters **`data_frame`** – The data frame to construct the data set from.

Returns

Return type The constructed data set.

`classmethod from_xml`(*xml*, *default_source*)
Load a ThermoML data set from an xml object.

Parameters

- **`xml`** (*`str`*) – The xml string to parse.
- **`default_source`** (`Source`) – The source to use if one cannot be parsed from the archive itself.

Returns The loaded ThermoML data set.

Return type *ThermoMLDataSet*

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*data_set*, *validate=True*)
Merge another data set into the current one.

Parameters

- **`data_set`** (`PhysicalPropertyDataSet`) – The secondary data set to merge into this one.
- **`validate`** (*`bool`*) – Whether to validate the other data set before merging.

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`property properties`
A list of all of the properties within this set.

Type tuple of PhysicalProperty

`properties_by_substance`(*substance*)

A generator which may be used to loop over all of the properties which were measured for a particular substance.

Parameters `substance` (`Substance`) – The substance of interest.

Returns

Return type generator of PhysicalProperty

`properties_by_type`(*property_type*)

A generator which may be used to loop over all of properties of a particular type, e.g. all "Density" properties.

Parameters `property_type` (*`str or type of PhysicalProperty`*) – The type of property of interest. This may either be the string class name of the property or the class type.

Returns

Return type generator of PhysicalProperty

`property property_types`

The types of property within this data set.

Type set of str

`property sources`

The sources from which the properties in this data set were gathered.

Type set of Source

`property substances`

The substances for which the properties in this data set were collected for.

Type set of Substance

`to_pandas`()

Converts a *PhysicalPropertyDataSet* to a *pandas.DataFrame* object with columns of

- 'Id'
- 'Temperature (K)'
- 'Pressure (kPa)'
- 'Phase'
- 'N Components'
- 'Component 1'
- 'Role 1'
- 'Mole Fraction 1'
- 'Exact Amount 1'
- …
- 'Component N'
- 'Role N'
- 'Mole Fraction N'
- 'Exact Amount N'
- '<Property 1> Value (<default unit>)'

- '<Property 1> Uncertainty / (<default unit>)'
- ...
- '<Property N> Value / (<default unit>)'
- '<Property N> Uncertainty / (<default unit>)'
- *'Source'*

where 'Component X' is a column containing the smiles representation of component X.

Returns The create data frame.

Return type pandas.DataFrame

validate()

Checks to ensure that all properties within the set are valid physical property object.

register_thermoml_property

openff.evaluator.datasets.thermoml.**register_thermoml_property**(*thermoml_string*, *supported_phases*, *property_class=None*, *conversion_function=None*)

A function used to map a property from the ThermoML archive to an internal *PhysicalProperty* object of the correct type.

This function takes either a specific class (e.g. *Density*) which maps directly to the specified *thermoml_string*, or a a function which maps a *ThermoMLProperty* into a *PhysicalProperty* allowing fuller control.

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.
- **property_class** (*type of PhysicalProperty, optional*) – The class associated with this physical property. This argument is mutually exclusive with the *conversion_function* argument.
- **conversion_function** (*function*) – A function which maps a *ThermoMLProperty* into a *PhysicalProperty*. This argument is mutually exclusive with the *property_class* argument.

thermoml_property

openff.evaluator.datasets.thermoml.**thermoml_property**(*thermoml_string*, *supported_phases*)

A decorator which wraps around the *register_thermoml_property* method.

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.

Taproom

TaproomDataSet	A dataset of host-guest binding affinity measurements which sources its data from the taproom package.
TaproomSource	Contains metadata about the source of a host-guest binding affinity measurement which was pulled from the `taproom` package.

TaproomDataSet

class openff.evaluator.datasets.taproom.**TaproomDataSet**(*host_codes: Optional[List[str]] = None*, *guest_codes: Optional[List[str]] = None*, *default_ionic_strength: Optional[openff.evaluator.utils.units.Quantity] = <Quantity(150, 'millimolar')>*, *negative_buffer_ion: str = '[Cl-]'*, *positive_buffer_ion: str = '[Na+]'*, *attach_apr_meta_data: bool = True*)

A dataset of host-guest binding affinity measurements which sources its data from the taproom package.

The loaded `HostGuestBindingAffinity` properties will also be optionally (enabled by default) initialized with the metadata required by the APR estimation workflow.

__init__(*host_codes: Optional[List[str]] = None*, *guest_codes: Optional[List[str]] = None*, *default_ionic_strength: Optional[openff.evaluator.utils.units.Quantity] = <Quantity(150, 'millimolar')>*, *negative_buffer_ion: str = '[Cl-]'*, *positive_buffer_ion: str = '[Na+]'*, *attach_apr_meta_data: bool = True*)

Parameters

- **host_codes** – The three letter codes of the host molecules to load from `taproom` If no list is provided, all hosts will be loaded.
- **guest_codes** – The three letter codes of the guest molecules to load from `taproom`. If no list is provided, all guests will be loaded.
- **default_ionic_strength** – The default ionic strength to use for measurements. The value specified in `taproom` will be ignored and this value used instead. If no value is provided, no buffer will be included.
- **negative_buffer_ion** – The SMILES pattern of the negative buffer ion to use. The value specified in `taproom` will be ignored and this value used instead.
- **positive_buffer_ion** – The SMILES pattern of the positive buffer ion to use. The value specified in `taproom` will be ignored and this value used instead.
- **attach_apr_meta_data** – Whether to add the metadata required for an APR based calculation using the `paprika` based workflow.

Methods

`__init__`([host_codes, guest_codes, ...])	**param host_codes** The three letter codes of the host molecules to load from `taproom`
`add_properties`(*physical_properties[, validate])	Adds a physical property to the data set.
`from_json`(file_path)	Create this object from a JSON file.
`from_pandas`(data_frame)	Constructs a data set object from a pandas `DataFrame` object.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(data_set[, validate])	Merge another data set into the current one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`properties_by_substance`(substance)	A generator which may be used to loop over all of the properties which were measured for a particular substance.
`properties_by_type`(property_type)	A generator which may be used to loop over all of properties of a particular type, e.g.
`to_pandas`()	Converts a *PhysicalPropertyDataSet* to a *pandas.DataFrame* object with columns of
`validate`()	Checks to ensure that all properties within the set are valid physical property object.

Attributes

`properties`	A list of all of the properties within this set.
`property_types`	The types of property within this data set.
`sources`	The sources from which the properties in this data set were gathered.
`substances`	The substances for which the properties in this data set were collected for.

add_properties(**physical_properties*, *validate=True*)

Adds a physical property to the data set.

Parameters

- **physical_properties** (`PhysicalProperty`) – The physical property to add.
- **validate** (*bool*) – Whether to validate the properties before adding them to the set.

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_pandas(*data_frame: pandas.core.frame.DataFrame*) → *openff.evaluator.datasets.datasets.PhysicalPropertyDataSet*

Constructs a data set object from a pandas `DataFrame` object.

Notes

- All physical properties are assumed to be source from experimental measurements.
- Currently this method onlu supports data frames containing properties which are built-in to the framework (e.g. Density).
- This method assumes the data frame has a structure identical to that produced by the `PhysicalPropertyDataSet.to_pandas` function.

Parameters **`data_frame`** – The data frame to construct the data set from.

Returns

Return type The constructed data set.

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*data_set*, *validate=True*)

Merge another data set into the current one.

Parameters

- **`data_set`** (`PhysicalPropertyDataSet`) – The secondary data set to merge into this one.
- **`validate`** (*`bool`*) – Whether to validate the other data set before merging.

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`property properties`

A list of all of the properties within this set.

Type tuple of PhysicalProperty

`properties_by_substance`(*substance*)

A generator which may be used to loop over all of the properties which were measured for a particular substance.

Parameters **`substance`** (`Substance`) – The substance of interest.

Returns

Return type generator of PhysicalProperty

`properties_by_type`(*property_type*)

A generator which may be used to loop over all of properties of a particular type, e.g. all “Density” properties.

Parameters `property_type` (*str or type of PhysicalProperty*) – The type of property of interest. This may either be the string class name of the property or the class type.

Returns

Return type generator of PhysicalProperty

property property_types
The types of property within this data set.

Type set of str

property sources
The sources from which the properties in this data set were gathered.

Type set of Source

property substances
The substances for which the properties in this data set were collected for.

Type set of Substance

to_pandas()
Converts a *PhysicalPropertyDataSet* to a *pandas.DataFrame* object with columns of

- 'Id'
- 'Temperature (K)'
- 'Pressure (kPa)'
- 'Phase'
- 'N Components'
- 'Component 1'
- 'Role 1'
- 'Mole Fraction 1'
- 'Exact Amount 1'
- …
- 'Component N'
- 'Role N'
- 'Mole Fraction N'
- 'Exact Amount N'
- '<Property 1> Value (<default unit>)'
- '<Property 1> Uncertainty / (<default unit>)'
- …
- '<Property N> Value / (<default unit>)'
- '<Property N> Uncertainty / (<default unit>)'
- *'Source'*

where 'Component X' is a column containing the smiles representation of component X.

Returns The create data frame.

Return type pandas.DataFrame

validate()

Checks to ensure that all properties within the set are valid physical property object.

TaproomSource

class openff.evaluator.datasets.taproom.**TaproomSource**(*doi=''*, *comment=''*, *technique=''*, *host_identifier=''*, *guest_identifier=''*)

Contains metadata about the source of a host-guest binding affinity measurement which was pulled from the `taproom` package.

__init__(*doi=''*, *comment=''*, *technique=''*, *host_identifier=''*, *guest_identifier=''*)

Constructs a new MeasurementSource object.

Parameters

- **doi** (*str*) – The DOI for the source
- **comment** (*str*) – A description of where the value came from in the source.
- **technique** (*str*) – The technique used to measure this value.
- **host_identifier** (*str*) – The unique three letter host identifier
- **guest_identifier** (*str*) – The unique three letter guest identifier

Methods

__init__([doi, comment, technique, ...])	Constructs a new MeasurementSource object.
from_json(file_path)	Create this object from a JSON file.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.

classmethod **from_json**(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod **parse_json**(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

Data Set Curation

CurationComponent	A base component for curation components which apply a particular operation (such as filtering or data conversion) to a data set.
CurationComponentSchema	A base class for schemas which specify how particular curation components should be applied to a data set.

CurationComponent

class openff.evaluator.datasets.curation.components.**CurationComponent**

A base component for curation components which apply a particular operation (such as filtering or data conversion) to a data set.

__init__()

Methods

__init__()	
apply()	Apply this curation component to a data set.

classmethod apply(*data_set:* openff.evaluator.datasets.datasets.PhysicalPropertyDataSet, *schema:* openff.evaluator.datasets.curation.components.components.CurationComponentSchema, *n_processes: int = 1*) → *openff.evaluator.datasets.datasets.PhysicalPropertyDataSet*

classmethod apply(*data_set: pandas.core.frame.DataFrame*, *schema:* openff.evaluator.datasets.curation.components.components.CurationComponentSchema, *n_processes: int = 1*) → pandas.core.frame.DataFrame

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

CurationComponentSchema

class openff.evaluator.datasets.curation.components.**CurationComponentSchema**(**args: Any*, ***kwargs: Any*)

A base class for schemas which specify how particular curation components should be applied to a data set.

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

CurationWorkflow	A convenience class for applying a set of curation components sequentially to a data set.
CurationWorkflowSchema	A schemas which encodes how a set of curation components should be applied sequentially to a data set.

CurationWorkflow

class openff.evaluator.datasets.curation.workflow.**CurationWorkflow**

A convenience class for applying a set of curation components sequentially to a data set.

__init__()

Methods

__init__()	
apply()	Apply each component of this curation workflow to an initial data set in sequence.

classmethod apply(*data_set:* openff.evaluator.datasets.datasets.PhysicalPropertyDataSet, *schema:* openff.evaluator.datasets.curation.workflow.CurationWorkflowSchema, *n_processes: int = 1*) → *openff.evaluator.datasets.datasets.PhysicalPropertyDataSet*

classmethod apply(*data_set: pandas.core.frame.DataFrame*, *schema:* openff.evaluator.datasets.curation.workflow.CurationWorkflowSchema, *n_processes: int = 1*) → pandas.core.frame.DataFrame

Apply each component of this curation workflow to an initial data set in sequence.

Parameters

- **data_set** – The data set to apply the workflow to. This may either be a data set object or it's pandas representation.
- **schema** – The schema which defines the components to apply.
- **n_processes** – The number of processes that each component is allowed to parallelize across.

Returns

Return type The data set which has had the curation workflow applied to it.

CurationWorkflowSchema

class `openff.evaluator.datasets.curation.workflow.`**`CurationWorkflowSchema`**(**args: Any*, ***kwargs: Any*)

A schemas which encodes how a set of curation components should be applied sequentially to a data set.

`__init__`(**args: Any*, ***kwargs: Any*) → None

Methods

`__init__`(*args, **kwargs)	

Attributes

`component_schemas`	

Filtering

`FilterDuplicatesSchema`	
`FilterDuplicates`	A component to remove duplicate data points (within a specified precision) from a data set.
`FilterByTemperatureSchema`	
`FilterByTemperature`	A component which will filter out data points which were measured outside of a specified temperature range
`FilterByPressureSchema`	
`FilterByPressure`	A component which will filter out data points which were measured outside of a specified pressure range.
`FilterByMoleFractionSchema`	
`FilterByMoleFraction`	A component which will filter out data points which were measured outside of a specified mole fraction range.
`FilterByRacemicSchema`	
`FilterByRacemic`	A component which will filter out data points which were measured for racemic mixtures.
`FilterByElementsSchema`	
`FilterByElements`	A component which will filter out data points which were measured for systems which contain specific elements.

continues on next page

Table 73 – continued from previous page

FilterByPropertyTypesSchema	
FilterByPropertyTypes	A component which will apply a filter which only retains properties of specified types.
FilterByStereochemistrySchema	
FilterByStereochemistry	A component which filters out data points measured for systems whereby the stereochemistry of a number of components is undefined.
FilterByChargedSchema	
FilterByCharged	A component which filters out data points measured for substances where any of the constituent components have a net non-zero charge.
FilterByIonicLiquidSchema	
FilterByIonicLiquid	A component which filters out data points measured for substances which contain or are classed as an ionic liquids.
FilterBySmilesSchema	
FilterBySmiles	A component which filters the data set so that it only contains either a specific set of smiles, or does not contain any of a set of specifically excluded smiles.
FilterBySmirksSchema	
FilterBySmirks	A component which filters a data set so that it only contains measurements made for molecules which contain (or don't) a set of chemical environments represented by SMIRKS patterns.
FilterByNComponentsSchema	
FilterByNComponents	A component which filters out data points measured for systems with specified number of components.
FilterBySubstancesSchema	
FilterBySubstances	A component which filters the data set so that it only contains properties measured for particular substances.
FilterByEnvironmentsSchema	
FilterByEnvironments	A component which filters a data set so that it only contains measurements made for substances which contain specific chemical environments.

FilterDuplicatesSchema

class openff.evaluator.datasets.curation.components.filtering.**FilterDuplicatesSchema**(**args: Any, **kwargs: Any*)

__init__(**args: Any, **kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

Attributes

mole_fraction_precision	
pressure_precision	
temperature_precision	
type	

FilterDuplicates

class openff.evaluator.datasets.curation.components.filtering.**FilterDuplicates**

A component to remove duplicate data points (within a specified precision) from a data set.

__init__()

Methods

__init__()	
apply(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod apply(*data_set, schema, n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterByTemperatureSchema

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterByTemperatureSchema`**(**args: Any*, ***kwargs: Any*)

`__init__`(**args: Any*, ***kwargs: Any*) → None

Methods

`__init__`(*args, **kwargs)	

Attributes

`maximum_temperature`
`minimum_temperature`
`type`

FilterByTemperature

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterByTemperature`**

A component which will filter out data points which were measured outside of a specified temperature range

`__init__`()

Methods

`__init__`()	
`apply`(data_set, schema[, n_processes])	Apply this curation component to a data set.

`classmethod` **`apply`**(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **`data_set`** – The data frame to apply the component to.
- **`schema`** – The schema which defines how this component should be applied.
- **`n_processes`** – The number of processes that this component is allowed to parallelize

across.

Returns

Return type The data set which has had the component applied to it.

FilterByPressureSchema

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterByPressureSchema`**`(`*`*args: Any, **kwargs: Any`*`)`

`__init__``(`*`*args: Any, **kwargs: Any`*`)` → None

Methods

`__init__`(*args, **kwargs)	

Attributes

`maximum_pressure`	
`minimum_pressure`	
`type`	

FilterByPressure

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterByPressure`**

A component which will filter out data points which were measured outside of a specified pressure range.

`__init__``()`

Methods

`__init__`()	
`apply`(data_set, schema[, n_processes])	Apply this curation component to a data set.

`classmethod` **`apply`**`(`*`data_set, schema, n_processes=1`*`)`

Apply this curation component to a data set.

Parameters

- **`data_set`** – The data frame to apply the component to.
- **`schema`** – The schema which defines how this component should be applied.

- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterByMoleFractionSchema

class openff.evaluator.datasets.curation.components.filtering.**FilterByMoleFractionSchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

Attributes

mole_fraction_ranges	
type	

FilterByMoleFraction

class openff.evaluator.datasets.curation.components.filtering.**FilterByMoleFraction**

A component which will filter out data points which were measured outside of a specified mole fraction range.

__init__()

Methods

__init__()	
apply(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod apply(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.

- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterByRacemicSchema

class openff.evaluator.datasets.curation.components.filtering.**FilterByRacemicSchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

Attributes

type	

FilterByRacemic

class openff.evaluator.datasets.curation.components.filtering.**FilterByRacemic**

A component which will filter out data points which were measured for racemic mixtures.

__init__()

Methods

__init__()	
apply(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod **apply**(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterByElementsSchema

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterByElementsSchema`**`(`*`*args: Any, **kwargs: Any`*`)`

`__init__``(`*`*args: Any, **kwargs: Any`*`)` → None

Methods

`__init__`(*args, **kwargs)	

Attributes

`allowed_elements`
`forbidden_elements`
`type`

FilterByElements

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterByElements`**

A component which will filter out data points which were measured for systems which contain specific elements.

`__init__``()`

Methods

`__init__`()	
`apply`(data_set, schema[, n_processes])	Apply this curation component to a data set.

`classmethod` **`apply`**`(`*`data_set, schema, n_processes=1`*`)`

Apply this curation component to a data set.

Parameters

- **`data_set`** – The data frame to apply the component to.
- **`schema`** – The schema which defines how this component should be applied.

- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterByPropertyTypesSchema

class openff.evaluator.datasets.curation.components.filtering.**FilterByPropertyTypesSchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

Attributes

n_components
property_types
strict
type

FilterByPropertyTypes

class openff.evaluator.datasets.curation.components.filtering.**FilterByPropertyTypes**

A component which will apply a filter which only retains properties of specified types.

__init__()

Methods

__init__()	
apply(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod **apply**(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterByStereochemistrySchema

class openff.evaluator.datasets.curation.components.filtering.**FilterByStereochemistrySchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

Attributes

type	

FilterByStereochemistry

class openff.evaluator.datasets.curation.components.filtering.**FilterByStereochemistry**

A component which filters out data points measured for systems whereby the stereochemistry of a number of components is undefined.

__init__()

Methods

__init__()	
apply(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod **apply**(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.

- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterByChargedSchema

class openff.evaluator.datasets.curation.components.filtering.**FilterByChargedSchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

Attributes

type	

FilterByCharged

class openff.evaluator.datasets.curation.components.filtering.**FilterByCharged**

A component which filters out data points measured for substances where any of the constituent components have a net non-zero charge.

__init__()

Methods

__init__()	
apply(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod **apply**(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.

- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterByIonicLiquidSchema

class openff.evaluator.datasets.curation.components.filtering.**FilterByIonicLiquidSchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

Attributes

type	

FilterByIonicLiquid

class openff.evaluator.datasets.curation.components.filtering.**FilterByIonicLiquid**

A component which filters out data points measured for substances which contain or are classed as an ionic liquids.

__init__()

Methods

__init__()	
apply(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod **apply**(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize

across.

Returns

Return type The data set which has had the component applied to it.

FilterBySmilesSchema

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterBySmilesSchema`**`(`*`*args: Any, **kwargs: Any`*`)`

`__init__``(`*`*args: Any, **kwargs: Any`*`)` → None

Methods

`__init__`(*args, **kwargs)	

Attributes

`allow_partial_inclusion`
`smiles_to_exclude`
`smiles_to_include`
`type`

FilterBySmiles

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterBySmiles`**

A component which filters the data set so that it only contains either a specific set of smiles, or does not contain any of a set of specifically excluded smiles.

`__init__``()`

Methods

`__init__`()	
`apply`(data_set, schema[, n_processes])	Apply this curation component to a data set.

`classmethod apply``(`*`data_set, schema, n_processes=1`*`)`

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterBySmirksSchema

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterBySmirksSchema`**`(*args: Any, **kwargs: Any)`

`__init__(*args: Any, **kwargs: Any) → None`

Methods

`__init__(*args, **kwargs)`	

Attributes

`allow_partial_inclusion`	
`smirks_to_exclude`	
`smirks_to_include`	
`type`	

FilterBySmirks

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterBySmirks`**

A component which filters a data set so that it only contains measurements made for molecules which contain (or don't) a set of chemical environments represented by SMIRKS patterns.

`__init__()`

Methods

`__init__`()	
`apply`(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod apply(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterByNComponentsSchema

class openff.evaluator.datasets.curation.components.filtering.**FilterByNComponentsSchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

`__init__`(*args, **kwargs)

Attributes

`n_components`
`type`

FilterByNComponents

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterByNComponents`**

A component which filters out data points measured for systems with specified number of components.

`__init__``()`

Methods

`__init__`()	
`apply`(data_set, schema[, n_processes])	Apply this curation component to a data set.

`classmethod apply`(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **`data_set`** – The data frame to apply the component to.
- **`schema`** – The schema which defines how this component should be applied.
- **`n_processes`** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterBySubstancesSchema

`class openff.evaluator.datasets.curation.components.filtering.`**`FilterBySubstancesSchema`**(**args: Any*, ***kwargs: Any*)

`__init__`(**args: Any*, ***kwargs: Any*) → None

Methods

`__init__`(*args, **kwargs)	

Attributes

substances_to_exclude
substances_to_include
type

FilterBySubstances

class openff.evaluator.datasets.curation.components.filtering.**FilterBySubstances**

A component which filters the data set so that it only contains properties measured for particular substances.

This method is similar to *filter_by_smiles*, however here we explicitly define the full substances compositions, rather than individual smiles which should either be included or excluded.

Examples

To filter the data set to only include measurements for pure methanol, pure benzene or an aqueous ethanol mix:

```
>>> schema = FilterBySubstancesSchema(
>>>     substances_to_include=[
>>>         ('CO',),
>>>         ('C1=CC=CC=C1',),
>>>         ('CCO', 'O')
>>>     ]
>>> )
```

To filter out measurements made for an aqueous mix of benzene:

```
>>> schema = FilterBySubstancesSchema(
>>>     substances_to_exclude=[('O', 'C1=CC=CC=C1')]
>>> )
```

__init__()

Methods

__init__()	
apply(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod apply(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.

- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FilterByEnvironmentsSchema

class openff.evaluator.datasets.curation.components.filtering.**FilterByEnvironmentsSchema**(**args: Any, **kwargs: Any*)

__init__(**args: Any, **kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

Attributes

at_least_one_environment	
environments	
per_component_environments	
strictly_specified_environments	
type	

FilterByEnvironments

class openff.evaluator.datasets.curation.components.filtering.**FilterByEnvironments**

A component which filters a data set so that it only contains measurements made for substances which contain specific chemical environments.

__init__()

Methods

`__init__`()	
`apply`(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod apply(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

FreeSolv

`ImportFreeSolvSchema`	
`ImportFreeSolv`	A component which will import the latest version of the FreeSolv data set from the GitHub repository where it is stored.

ImportFreeSolvSchema

class openff.evaluator.datasets.curation.components.freesolv.**ImportFreeSolvSchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

`__init__`(*args, **kwargs)	

Attributes

`type`

ImportFreeSolv

`class openff.evaluator.datasets.curation.components.freesolv.`**`ImportFreeSolv`**

A component which will import the latest version of the FreeSolv data set from the GitHub repository where it is stored.

`__init__``()`

Methods

`__init__`()	
`apply`(data_set, schema[, n_processes])	Apply this curation component to a data set.

`classmethod` **`apply`**(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **`data_set`** – The data frame to apply the component to.
- **`schema`** – The schema which defines how this component should be applied.
- **`n_processes`** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

ThermoML

`ImportThermoMLDataSchema`	
`ImportThermoMLData`	A component which will import all supported data from the NIST ThermoML archive for (optionally) specified journals.

ImportThermoMLDataSchema

`class openff.evaluator.datasets.curation.components.thermoml.`**`ImportThermoMLDataSchema`**(**args: Any*, ***kwargs: Any*)

`__init__`(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

Attributes

cache_file_name	
retain_uncertainties	
root_archive_url	
type	

ImportThermoMLData

class openff.evaluator.datasets.curation.components.thermoml.**ImportThermoMLData**

A component which will import all supported data from the NIST ThermoML archive for (optionally) specified journals.

__init__()

Methods

__init__()	
apply(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod **apply**(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

Data Point Selection

SelectSubstancesSchema

continues on next page

Table 127 – continued from previous page

SelectSubstances	A component for selecting a specified number data points which were measured for systems containing a specified set of chemical functionalities.
SelectDataPointsSchema	
SelectDataPoints	A component for selecting a set of data points which are measured as close as possible to a particular set of states.
State	
TargetState	
FingerPrintType	An enumeration.

SelectSubstancesSchema

class openff.evaluator.datasets.curation.components.selection.**SelectSubstancesSchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)	

Attributes

finger_print_type	
n_per_environment	
per_property	
substances_to_exclude	
target_environments	
type	

SelectSubstances

class openff.evaluator.datasets.curation.components.selection.**SelectSubstances**

A component for selecting a specified number data points which were measured for systems containing a specified set of chemical functionalities.

__init__()

Methods

__init__()	
apply(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod **apply**(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

SelectDataPointsSchema

class openff.evaluator.datasets.curation.components.selection.**SelectDataPointsSchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)

Attributes

`target_states`	
`type`	

SelectDataPoints

class `openff.evaluator.datasets.curation.components.selection.`**`SelectDataPoints`**

A component for selecting a set of data points which are measured as close as possible to a particular set of states.

The points will be chosen so as to try and maximise the number of properties measured at the same condition (e.g. ideally we would have a data point for each property at T=298.15 and p=1atm) as this will maximise the chances that we can extract all properties from a single simulation.

`__init__`()

Methods

`__init__`()	
`apply`(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod apply(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **`data_set`** – The data frame to apply the component to.
- **`schema`** – The schema which defines how this component should be applied.
- **`n_processes`** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

State

class `openff.evaluator.datasets.curation.components.selection.`**`State`**(**args: Any*, ***kwargs: Any*)

`__init__`(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)

Attributes

`mole_fractions`
`pressure`
`temperature`

TargetState

class `openff.evaluator.datasets.curation.components.selection.`**`TargetState`**(**args: Any, **kwargs: Any*)

`__init__`(**args: Any, **kwargs: Any*) → None

Methods

__init__(*args, **kwargs)

Attributes

`property_types`
`property_types_validator`
`states`

FingerPrintType

class openff.evaluator.datasets.curation.components.selection.**FingerPrintType**(*value*)

An enumeration.

__init__()

Attributes

Tree
MACCS166

Data Conversion

ConvertExcessDensityDataSchema	
ConvertExcessDensityData	A component for converting binary mass density data to excess molar volume data and vice versa where pure density data measured for the components is available.

ConvertExcessDensityDataSchema

class openff.evaluator.datasets.curation.components.conversion.**ConvertExcessDensityDataSchema**(**args: Any*, ***kwargs: Any*)

__init__(**args: Any*, ***kwargs: Any*) → None

Methods

__init__(*args, **kwargs)

Attributes

pressure_precision
temperature_precision
type

ConvertExcessDensityData

class openff.evaluator.datasets.curation.components.conversion.**ConvertExcessDensityData**

A component for converting binary mass density data to excess molar volume data and vice versa where pure density data measured for the components is available.

Notes

This protocol may result in duplicate data points being generated. It is recommended to apply the de-duplication filter after this component has been applied.

`__init__`()

Methods

`__init__`()	
`apply`(data_set, schema[, n_processes])	Apply this curation component to a data set.

classmethod apply(*data_set*, *schema*, *n_processes=1*)

Apply this curation component to a data set.

Parameters

- **data_set** – The data frame to apply the component to.
- **schema** – The schema which defines how this component should be applied.
- **n_processes** – The number of processes that this component is allowed to parallelize across.

Returns

Return type The data set which has had the component applied to it.

2.32.5 Force Field API

ForceFieldSource	A helper object to define the source of a force field and any associated meta data, such as version, file paths, or generation options.
SmirnoffForceFieldSource	A wrapper around force fields based on the SMIRks Native Open Force Field (SMIRNOFF) specification.
TLeapForceFieldSource	A wrapper around Amber force fields which may be applied via the *tleap* software package.
LigParGenForceFieldSource	A wrapper and the OPLSAAM force field which can be applied via the LigParGen server.

ForceFieldSource

class openff.evaluator.forcefield.**ForceFieldSource**

A helper object to define the source of a force field and any associated meta data, such as version, file paths, or generation options.

Notes

It is likely that this class and classes based off of it will not be permanent fixtures of the framework, but rather will exist until the force fields can be stored in a uniform format / object model.

`__init__`()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

SmirnoffForceFieldSource

class `openff.evaluator.forcefield.`**`SmirnoffForceFieldSource`**(*inner_xml=None*)

A wrapper around force fields based on the SMIRks Native Open Force Field (SMIRNOFF) specification.

`__init__`(*inner_xml=None*)

Constructs a new SmirnoffForceFieldSource object

Parameters **`inner_xml`** (*`str`, `optional`*) – A string containing the xml representation of the force field.

Methods

`__init__`([inner_xml])	Constructs a new SmirnoffForceFieldSource object
`from_json`(file_path)	Create this object from a JSON file.
`from_object`(force_field)	Creates a new *SmirnoffForceFieldSource* from an existing *ForceField* object
`from_path`(file_path)	Creates a new *SmirnoffForceFieldSource* from the file path to a *ForceField* object.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`to_force_field`()	Returns the SMIRNOFF force field created from this source.

`to_force_field`()

Returns the SMIRNOFF force field created from this source.

Returns The created force field.

Return type openff.toolkit.typing.engines.smirnoff.ForceField

classmethod **`from_object`**(*force_field*)

Creates a new *SmirnoffForceFieldSource* from an existing *ForceField* object

Notes

All cosmetic attributes will be discarded.

Parameters **`force_field`** (*`openff.toolkit.typing.engines.smirnoff.ForceField`*) – The existing force field.

Returns The created object.

Return type *SmirnoffForceFieldSource*

classmethod **`from_path`**(*file_path*)

Creates a new *SmirnoffForceFieldSource* from the file path to a *ForceField* object.

Notes

All cosmetic attributes will be discarded.

Parameters **file_path** (*str*) – The file path to the force field object. This may also be the name of a file which can be loaded via an entry point.

Returns The created object.

Return type *SmirnoffForceFieldSource*

`classmethod` **from_json**(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod` **parse_json**(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

TLeapForceFieldSource

`class` `openff.evaluator.forcefield.`**TLeapForceFieldSource**(*leap_source='leaprc.gaff2'*, *cutoff=<Quantity(9.0, 'angstrom')>*)

A wrapper around Amber force fields which may be applied via the *tleap* software package.

Notes

Currently this only supports force fields which are installed alongside *tleap*.

__init__(*leap_source='leaprc.gaff2'*, *cutoff=<Quantity(9.0, 'angstrom')>*)

Constructs a new TLeapForceFieldSource object

Parameters

- **leap_source** (*str*) – The parameter file which should be sourced by *leap* when applying the force field. Currently only *'leaprc.gaff'* and *'leaprc.gaff2'* are supported.
- **cutoff** (`openff.evaluator.unit.Quantity`) – The non-bonded interaction cutoff.

Examples

To create a source for the GAFF force field with tip3p water:

```
>>> amber_gaff_source = TLeapForceFieldSource('leaprc.gaff')
```

To create a source for the GAFF 2 force field with tip3p water:

```
>>> amber_gaff_2_source = TLeapForceFieldSource('leaprc.gaff2')
```

Methods

`__init__`([leap_source, cutoff])	Constructs a new TLeapForceFieldSource object
`from_json`(file_path)	Create this object from a JSON file.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.

Attributes

`cutoff`	The non-bonded interaction cutoff.
`leap_source`	The parameter file which should be sourced by *leap* when applying the force field.

property leap_source
The parameter file which should be sourced by *leap* when applying the force field.

Type list of str

property cutoff
The non-bonded interaction cutoff.

Type openff.evaluator.unit.Quantity

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (`str`) – The path to load the JSON from.

Returns The parsed class.

Return type cls

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (`str, optional`) – The (optional) file path to save the JSON file to.
- **format** (`bool`) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

LigParGenForceFieldSource

class openff.evaluator.forcefield.**LigParGenForceFieldSource**(*preferred_charge_model=ChargeModel.CM1A_1_14_LB*
cutoff=<Quantity(9.0, 'angstrom')>,
request_url='', *download_url=''*)

A wrapper and the OPLSAAM force field which can be applied via the LigParGen server.

References

[1] **Potential energy functions for atomic-level simulations of water and organic and** biomolecular systems. Jorgensen, W. L.; Tirado-Rives, J. Proc. Nat. Acad. Sci. USA 2005, 102, 6665-6670

[2] **1.14*CM1A-LBCC: Localized Bond-Charge Corrected CM1A Charges for Condensed-Phase** Simulations. Dodda, L. S.; Vilseck, J. Z.; Tirado-Rives, J.; Jorgensen, W. L. J. Phys. Chem. B, 2017, 121 (15), pp 3864-3870

[3] **LigParGen web server: An automatic OPLS-AA parameter generator for organic ligands.** Dodda, L. S.;Cabeza de Vaca, I.; Tirado-Rives, J.; Jorgensen, W. L. Nucleic Acids Research, Volume 45, Issue W1, 3 July 2017, Pages W331-W336

__init__(*preferred_charge_model=ChargeModel.CM1A_1_14_LBCC*, *cutoff=<Quantity(9.0, 'angstrom')>*, *request_url=''*, *download_url=''*)

Constructs a new LigParGenForceFieldSource object

Parameters

- **preferred_charge_model** (ChargeModel) – The preferred charge model to apply. In some cases the preferred charge model may not be applicable (e.g. 1.14*CM1A-LBCC may only be applied to neutral molecules) and so another model may be applied in its place.
- **cutoff** (*openff.evaluator.unit.Quantity*) – The non-bonded interaction cutoff.
- **request_url** (*str*) – The URL of the LIGPARGEN server file to send the parametrization to request to.
- **download_url** (*str*) – The URL of the LIGPARGEN server file to download the results of a request from.

Methods

__init__([preferred_charge_model, cutoff, ...])	Constructs a new LigParGenForceFieldSource object
from_json(file_path)	Create this object from a JSON file.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.

Attributes

cutoff	The non-bonded interaction cutoff.
download_url	The URL of the LIGPARGEN server file to download the results of a request from.
preferred_charge_model	The preferred charge model to apply.
request_url	The URL of the LIGPARGEN server file to send the parametrization to request to.

class ChargeModel(*value*)
An enumeration.

property preferred_charge_model
The preferred charge model to apply. In some cases the preferred charge model may not be applicable (e.g. 1.14*CM1A-LBCC may only be applied to neutral molecules) and so another model may be applied in its place.

Type *ChargeModel*

property cutoff
The non-bonded interaction cutoff.

Type openff.evaluator.unit.Quantity

property request_url
The URL of the LIGPARGEN server file to send the parametrization to request to.

Type str

property download_url
The URL of the LIGPARGEN server file to download the results of a request from.

Type str

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters file_path (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

Gradient Estimation

ParameterGradientKey	
ParameterGradient	

ParameterGradientKey

class openff.evaluator.forcefield.**ParameterGradientKey**(*tag=None*, *smirks=None*, *attribute=None*)

__init__(*tag=None*, *smirks=None*, *attribute=None*)

Methods

__init__([tag, smirks, attribute])	

Attributes

attribute	
smirks	
tag	

ParameterGradient

class openff.evaluator.forcefield.**ParameterGradient**(*key=None*, *value=None*)

__init__(*key=None*, *value=None*)

Methods

__init__([key, value])	

Attributes

key	
value	

2.32.6 Calculation Layers API

CalculationLayer	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.
CalculationLayerResult	The result of attempting to estimate a property using a *CalculationLayer*.
CalculationLayerSchema	A schema which encodes the options that a *CalculationLayer* should use when estimating a given class of physical properties.
calculation_layer	A decorator which registers a class as being a calculation layer which may be used in property calculations.
register_calculation_layer	Registers a class as being a calculation layer which may be used in property calculations.
register_calculation_schema	Registers the default calculation schema to use when estimating a class of properties (e.g.

CalculationLayer

class openff.evaluator.layers.**CalculationLayer**

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.

__init__()

Methods

__init__()	
required_schema_type()	Returns the type of *CalculationLayerSchema* required by this layer.
schedule_calculation(calculation_backend, ...)	Submit the proposed calculation to the backend of choice.

abstract classmethod required_schema_type()

Returns the type of *CalculationLayerSchema* required by this layer.

Returns The required schema type.

Return type type of CalculationLayerSchema

classmethod schedule_calculation(*calculation_backend*, *storage_backend*, *layer_directory*, *batch*, *callback*, *synchronous=False*)

Submit the proposed calculation to the backend of choice.

Parameters

- **calculation_backend** (CalculationBackend) – The backend to the submit the calculations to.
- **storage_backend** (StorageBackend) – The backend used to store / retrieve data from previous calculations.
- **layer_directory** (*str*) – The directory in which to store all temporary calculation data from this layer.
- **batch** (Batch) – The batch of properties to estimate with the layer.
- **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.

CalculationLayerResult

class openff.evaluator.layers.**CalculationLayerResult**

The result of attempting to estimate a property using a *CalculationLayer*.

__init__()

Methods

__init__()	
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.

continues on next page

Table 157 – continued from previous page

validate([attribute_type])	Validate the values of the attributes.

Attributes

data_to_store	Paths to the data objects to store.
exceptions	Any exceptions raised by the layer while estimating the property.
physical_property	The estimated property (if the layer was successful).

physical_property
The estimated property (if the layer was successful). The default value of this attribute is not set. This attribute is *optional*.

Type *PhysicalProperty*

data_to_store
Paths to the data objects to store. The default value of this attribute is `[]`.

Type list

exceptions
Any exceptions raised by the layer while estimating the property. The default value of this attribute is `[]`.

Type list

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

CalculationLayerSchema

class openff.evaluator.layers.**CalculationLayerSchema**

A schema which encodes the options that a *CalculationLayer* should use when estimating a given class of physical properties.

__init__()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`absolute_tolerance`	The absolute uncertainty that the property should be estimated to within.
`relative_tolerance`	The relative uncertainty that the property should be estimated to within, i.e *relative_tolerance * measured_property.uncertainty*.

absolute_tolerance

The absolute uncertainty that the property should be estimated to within. This attribute is mutually exclusive with the *relative_tolerance* attribute. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

relative_tolerance

The relative uncertainty that the property should be estimated to within, i.e *relative_tolerance * measured_property.uncertainty*. This attribute is mutually exclusive with the *absolute_tolerance* attribute. The default value of this attribute is not set. This attribute is *optional*.

Type float

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

calculation_layer

`openff.evaluator.layers.calculation_layer()`
A decorator which registers a class as being a calculation layer which may be used in property calculations.

register_calculation_layer

`openff.evaluator.layers.`**`register_calculation_layer`**(*layer_class*)

Registers a class as being a calculation layer which may be used in property calculations.

Parameters **`layer_class`** (*`type of CalculationLayer`*) – The calculation layer to register.

register_calculation_schema

`openff.evaluator.layers.`**`register_calculation_schema`**(*property_class*, *layer_class*, *schema*)

Registers the default calculation schema to use when estimating a class of properties (e.g. *Density*) with a specific calculation layer (e.g. the *SimulationLayer*).

Parameters

- **`property_class`** (*`type of PhysicalProperty`*) – The class of properties to associate with the specified *calculation_layer* and *property_class*.
- **`layer_class`** (*`type of CalculationLayer`*) – The calculation layer to associate the schema with.
- **`schema`** (`CalculationLayerSchema` *`or Callable[[`*`CalculationLayerSchema`*`],`* `CalculationLayerSchema`*`]`*) – Either the calculation schema to use, or a function which will create the schema from an existing CalculationLayerSchema.

Built-in Calculation Layers

`WorkflowCalculationLayer`	An calculation layer which uses the built-in workflow framework to estimate sets of physical properties.
`WorkflowCalculationSchema`	A schema which encodes the options and the workflow schema that a *CalculationLayer* should use when estimating a given class of physical properties using the built-in workflow framework.

WorkflowCalculationLayer

`class openff.evaluator.layers.workflow.`**`WorkflowCalculationLayer`**

An calculation layer which uses the built-in workflow framework to estimate sets of physical properties.

`__init__`()

Methods

`__init__`()	
`required_schema_type`()	Returns the type of *CalculationLayerSchema* required by this layer.
`schedule_calculation`(calculation_backend, ...)	Submit the proposed calculation to the backend of choice.
`workflow_to_layer_result`(queued_properties, ...)	Converts a list of *WorkflowResult* to a list of *CalculationLayerResult* objects.

static workflow_to_layer_result(*queued_properties*, *provenance*, *workflow_results*, **_)

Converts a list of *WorkflowResult* to a list of *CalculationLayerResult* objects.

Parameters

- **queued_properties** (*list of PhysicalProperty*) – The properties being estimated by this layer
- **provenance** (*dict of str and str*) – The provenance of each property.
- **workflow_results** (*list of WorkflowResult*) – The results of each workflow.

Returns The calculation layer result objects.

Return type list of CalculationLayerResult

abstract classmethod required_schema_type()

Returns the type of *CalculationLayerSchema* required by this layer.

Returns The required schema type.

Return type type of CalculationLayerSchema

classmethod schedule_calculation(*calculation_backend*, *storage_backend*, *layer_directory*, *batch*, *callback*, *synchronous=False*)

Submit the proposed calculation to the backend of choice.

Parameters

- **calculation_backend** (CalculationBackend) – The backend to the submit the calculations to.
- **storage_backend** (StorageBackend) – The backend used to store / retrieve data from previous calculations.
- **layer_directory** (*str*) – The directory in which to store all temporary calculation data from this layer.
- **batch** (Batch) – The batch of properties to estimate with the layer.
- **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.

WorkflowCalculationSchema

class openff.evaluator.layers.workflow.**WorkflowCalculationSchema**

A schema which encodes the options and the workflow schema that a *CalculationLayer* should use when estimating a given class of physical properties using the built-in workflow framework.

__init__()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`absolute_tolerance`	The absolute uncertainty that the property should be estimated to within.
`relative_tolerance`	The relative uncertainty that the property should be estimated to within, i.e *relative_tolerance * measured_property.uncertainty*.
`workflow_schema`	The workflow schema to use when estimating properties.

workflow_schema
The workflow schema to use when estimating properties. The default value of this attribute is not set and must be set by the user..

Type *WorkflowSchema*

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (`type of Attribute, optional`) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

absolute_tolerance
The absolute uncertainty that the property should be estimated to within. This attribute is mutually exclusive with the *relative_tolerance* attribute. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (`str`) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (`type of Attribute, optional`) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`relative_tolerance`
The relative uncertainty that the property should be estimated to within, i.e *relative_tolerance* * *measured_property.uncertainty*. This attribute is mutually exclusive with the *absolute_tolerance* attribute. The default value of this attribute is not set. This attribute is *optional*.

Type float

SimulationLayer	A calculation layer which employs molecular simulation to estimate sets of physical properties.
SimulationSchema	A schema which encodes the options and the workflow schema that the *SimulationLayer* should use when estimating a given class of physical properties using the built-in workflow framework.

SimulationLayer

`class openff.evaluator.layers.simulation.SimulationLayer`
A calculation layer which employs molecular simulation to estimate sets of physical properties.

`__init__()`

Methods

__init__()	
required_schema_type()	Returns the type of *CalculationLayerSchema* required by this layer.
schedule_calculation(calculation_backend, ...)	Submit the proposed calculation to the backend of choice.
workflow_to_layer_result(queued_properties, ...)	Converts a list of *WorkflowResult* to a list of *CalculationLayerResult* objects.

classmethod required_schema_type()

Returns the type of *CalculationLayerSchema* required by this layer.

Returns The required schema type.

Return type type of CalculationLayerSchema

classmethod schedule_calculation(*calculation_backend*, *storage_backend*, *layer_directory*, *batch*, *callback*, *synchronous=False*)

Submit the proposed calculation to the backend of choice.

Parameters

- **calculation_backend** (CalculationBackend) – The backend to the submit the calculations to.
- **storage_backend** (StorageBackend) – The backend used to store / retrieve data from previous calculations.
- **layer_directory** (*str*) – The directory in which to store all temporary calculation data from this layer.
- **batch** (Batch) – The batch of properties to estimate with the layer.
- **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.

static workflow_to_layer_result(*queued_properties*, *provenance*, *workflow_results*, ***_*)

Converts a list of *WorkflowResult* to a list of *CalculationLayerResult* objects.

Parameters

- **queued_properties** (*list of PhysicalProperty*) – The properties being estimated by this layer
- **provenance** (*dict of str and str*) – The provenance of each property.
- **workflow_results** (*list of WorkflowResult*) – The results of each workflow.

Returns The calculation layer result objects.

Return type list of CalculationLayerResult

SimulationSchema

class openff.evaluator.layers.simulation.**SimulationSchema**

A schema which encodes the options and the workflow schema that the *SimulationLayer* should use when estimating a given class of physical properties using the built-in workflow framework.

__init__()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`absolute_tolerance`	The absolute uncertainty that the property should be estimated to within.
`relative_tolerance`	The relative uncertainty that the property should be estimated to within, i.e *relative_tolerance * measured_property.uncertainty*.
`workflow_schema`	The workflow schema to use when estimating properties.

`absolute_tolerance`

The absolute uncertainty that the property should be estimated to within. This attribute is mutually exclusive with the *relative_tolerance* attribute. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **`file_path`** (`str`) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (`type of Attribute, optional`) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (`str, optional`) – The (optional) file path to save the JSON file to.
- **`format`** (`bool`) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

relative_tolerance

The relative uncertainty that the property should be estimated to within, i.e *relative_tolerance * measured_property.uncertainty*. This attribute is mutually exclusive with the *absolute_tolerance* attribute. The default value of this attribute is not set. This attribute is *optional*.

Type float

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

workflow_schema

The workflow schema to use when estimating properties. The default value of this attribute is not set and must be set by the user..

Type *WorkflowSchema*

ReweightingLayer	A *CalculationLayer* which attempts to 'reweight' cached simulation data to evaluate the values of properties at states which have not previously been simulated directly, but where simulations at similar states have been run previously.
ReweightingSchema	A schema which encodes the options and the workflow schema that the *SimulationLayer* should use when estimating a given class of physical properties using the built-in workflow framework.
default_storage_query	Return the default query to use when retrieving cached simulation

ReweightingLayer

class openff.evaluator.layers.reweighting.**ReweightingLayer**

A *CalculationLayer* which attempts to 'reweight' cached simulation data to evaluate the values of properties at states which have not previously been simulated directly, but where simulations at similar states have been run previously.

__init__()

Methods

__init__()	
required_schema_type()	Returns the type of *CalculationLayerSchema* required by this layer.
schedule_calculation(calculation_backend, ...)	Submit the proposed calculation to the backend of choice.
workflow_to_layer_result(queued_properties, ...)	Converts a list of *WorkflowResult* to a list of *CalculationLayerResult* objects.

classmethod required_schema_type()

Returns the type of *CalculationLayerSchema* required by this layer.

Returns The required schema type.

Return type type of CalculationLayerSchema

classmethod schedule_calculation(*calculation_backend*, *storage_backend*, *layer_directory*, *batch*, *callback*, *synchronous=False*)

Submit the proposed calculation to the backend of choice.

Parameters

- **calculation_backend** (CalculationBackend) – The backend to the submit the calculations to.
- **storage_backend** (StorageBackend) – The backend used to store / retrieve data from previous calculations.
- **layer_directory** (*str*) – The directory in which to store all temporary calculation data from this layer.
- **batch** (Batch) – The batch of properties to estimate with the layer.
- **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.

static workflow_to_layer_result(*queued_properties*, *provenance*, *workflow_results*, ***_*)

Converts a list of *WorkflowResult* to a list of *CalculationLayerResult* objects.

Parameters

- **queued_properties** (*list of PhysicalProperty*) – The properties being estimated by this layer
- **provenance** (*dict of str and str*) – The provenance of each property.
- **workflow_results** (*list of WorkflowResult*) – The results of each workflow.

Returns The calculation layer result objects.

Return type list of CalculationLayerResult

ReweightingSchema

class `openff.evaluator.layers.reweighting.`**`ReweightingSchema`**

A schema which encodes the options and the workflow schema that the *SimulationLayer* should use when estimating a given class of physical properties using the built-in workflow framework.

`__init__()`

Methods

`__init__()`	
`from_json(file_path)`	Create this object from a JSON file.
`get_attributes([attribute_type])`	Returns all attributes of a specific *attribute_type*.
`json([file_path, format])`	Creates a JSON representation of this class.
`parse_json(string_contents)`	Parses a typed json string into the corresponding class structure.
`validate([attribute_type])`	Validate the values of the attributes.

Attributes

`absolute_tolerance`	The absolute uncertainty that the property should be estimated to within.
`maximum_data_points`	The maximum number of data points to include as part of the multi-state reweighting calculations.
`relative_tolerance`	The relative uncertainty that the property should be estimated to within, i.e *relative_tolerance * measured_property.uncertainty*.
`storage_queries`	The queries to perform when retrieving data for each of the components in the system from the storage backend.
`temperature_cutoff`	The maximum difference between the target temperature and the temperature at which cached data was collected to.
`workflow_schema`	The workflow schema to use when estimating properties.

`storage_queries`

The queries to perform when retrieving data for each of the components in the system from the storage backend. The keys of this dictionary will correspond to the metadata keys made available to the workflow system.

Type dict

`maximum_data_points`

The maximum number of data points to include as part of the multi-state reweighting calculations. If zero, no cap will be applied. The default value of this attribute is 4.

Type int

`temperature_cutoff`

The maximum difference between the target temperature and the temperature at which cached data was collected to. Data collected for temperatures outside of this cutoff will be ignored. The default value of this attribute is `5.0 K`.

Type Quantity

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`absolute_tolerance`
The absolute uncertainty that the property should be estimated to within. This attribute is mutually exclusive with the *relative_tolerance* attribute. The default value of this attribute is not set. This attribute is *optional*.

Type Quantity

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`relative_tolerance`
The relative uncertainty that the property should be estimated to within, i.e *relative_tolerance * measured_property.uncertainty*. This attribute is mutually exclusive with the *absolute_tolerance* attribute. The default value of this attribute is not set. This attribute is *optional*.

Type float

workflow_schema
The workflow schema to use when estimating properties. The default value of this attribute is not set and must be set by the user..

Type *WorkflowSchema*

default_storage_query

`openff.evaluator.layers.reweighting.`**`default_storage_query`**`()`

Return the default query to use when retrieving cached simulation data from the storage backend.

Currently this query will search for data for the full substance of interest in the liquid phase.

Returns A single query with a key of *"full_system_data"*.

Return type dict of str and SimulationDataQuery

2.32.7 Calculation Backends API

`CalculationBackend`	An abstract base representation of an openff-evaluator calculation 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.

CalculationBackend

class `openff.evaluator.backends.`**`CalculationBackend`**(*number_of_workers=1*, *resources_per_worker=None*)

An abstract base representation of an openff-evaluator calculation 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* across 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=None*)
Constructs a new CalculationBackend 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`, *optional*) – The number of resources to request per worker.

Methods

`__init__`([number_of_workers, ...])	Constructs a new CalculationBackend 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.

Attributes

`started`	Returns whether this backend has been started yet.

property started
Returns whether this backend has been started yet.

Type bool

start()
Start the calculation backend.

abstract stop()
Stop the calculation backend.

abstract 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 openff.evaluator.backends.**ComputeResources**(*number_of_threads=1*, *number_of_gpus=0*, *preferred_gpu_toolkit=GPUToolkit.CUDA*)
An object which stores how many of each type of computational resource (threads or gpu's) is available to a calculation worker.

__init__(*number_of_threads=1*, *number_of_gpus=0*, *preferred_gpu_toolkit=GPUToolkit.CUDA*)
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.
`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(*value*)
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 openff.evaluator.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** (*openmm.unit.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 openmm.unit.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(*value*)

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.
BaseDaskJobQueueBackend	An openff-evaluator backend which uses a *dask_jobqueue.JobQueueCluster* object to run calculations within an existing HPC queuing system.
DaskLocalCluster	An openff-evaluator backend which uses a *dask LocalCluster* object to run calculations on a single machine.
DaskLSFBackend	An openff-evaluator backend which uses a *dask_jobqueue.LSFCluster* object to run calculations within an existing LSF queue.
DaskPBSBackend	An openff-evaluator backend which uses a *dask_jobqueue.PBSCluster* object to run calculations within an existing PBS queue.

BaseDaskBackend

class openff.evaluator.backends.dask.**BaseDaskBackend**(*number_of_workers=1*, *resources_per_worker=<openff.evaluator.backends.backends.Comp*
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=<openff.evaluator.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.

Attributes

`started`	Returns whether this backend has been started yet.

start()
Start the calculation backend.

stop()
Stop the calculation backend.

property started
Returns whether this backend has been started yet.

Type bool

abstract 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

BaseDaskJobQueueBackend

class openff.evaluator.backends.dask.**BaseDaskJobQueueBackend**(*minimum_number_of_workers=1*, *maximum_number_of_workers=1*, *re-sources_per_worker=<openff.evaluator.backends.back* *object>*, *queue_name='default'*, *setup_script_commands=None*, *extra_script_options=None*, *adaptive_interval='10000ms'*, *disable_nanny_process=False*, *cluster_type=None*, *adaptive_class=None*)

An openff-evaluator backend which uses a *dask_jobqueue.JobQueueCluster* object to run calculations within an existing HPC queuing system.

See also:

`dask_jobqueue.JobQueueCluster`

__init__(*minimum_number_of_workers=1*, *maximum_number_of_workers=1*, *resources_per_worker=<openff.evaluator.backends.backends.QueueWorkerResources object>*, *queue_name='default'*, *setup_script_commands=None*, *extra_script_options=None*, *adaptive_interval='10000ms'*, *disable_nanny_process=False*, *cluster_type=None*, *adaptive_class=None*)
Constructs a new BaseDaskJobQueueBackend 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.
- **adaptive_class** (class of type *distributed.deploy.AdaptiveCore*, optional) – An optional class to pass to dask to use for its adaptive scaling handling. This is mainly exposed to allow easily working around certain dask bugs / quirks.

Methods

__init__([minimum_number_of_workers, ...])	Constructs a new BaseDaskJobQueueBackend object
job_script()	Returns the job script that dask will use to submit workers.
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.

Attributes

started	Returns whether this backend has been started yet.

job_script()

Returns the job script that dask will use to submit workers. The backend must be started before calling this function.

Returns

Return type str

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

property started
Returns whether this backend has been started yet.

Type bool

stop()
Stop the calculation backend.

DaskLocalCluster

class `openff.evaluator.backends.dask.`**DaskLocalCluster**(*number_of_workers=1*, *resources_per_worker=<openff.evaluator.backends.backends.Com object>*)

An openff-evaluator 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=<openff.evaluator.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.

Attributes

started	Returns whether this backend has been started yet.

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

property started
Returns whether this backend has been started yet.

Type bool

stop()
Stop the calculation backend.

DaskLSFBackend

class `openff.evaluator.backends.dask.`**`DaskLSFBackend`**(*minimum_number_of_workers=1*, *maximum_number_of_workers=1*, *resources_per_worker=<openff.evaluator.backends.backends.QueueW*
object>, *queue_name='default'*, *setup_script_commands=None*, *extra_script_options=None*, *adaptive_interval='10000ms'*, *disable_nanny_process=False*, *adaptive_class=None*)

An openff-evaluator backend which uses a *dask_jobqueue.LSFCluster* object to run calculations within an existing LSF queue.

See also:

`dask_jobqueue.LSFCluster`, *`DaskPBSBackend`*

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

Constructs a new DaskLSFBackend object

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 openff.evaluator.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.
```

(continues on next page)

(continued from previous page)

```
>>> from openff.evaluator.backends.dask import DaskLSFBackend
>>>
>>> lsf_backend = DaskLSFBackend(minimum_number_of_workers=1,
>>>                              maximum_number_of_workers=10,
>>>                              resources_per_worker=resources,
>>>                              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
job_script()	Returns the job script that dask will use to submit workers.
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.

Attributes

started	Returns whether this backend has been started yet.

job_script()
Returns the job script that dask will use to submit workers. The backend must be started before calling this function.

Returns

Return type str

start()
Start the calculation backend.

property started
Returns whether this backend has been started yet.

Type bool

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

DaskPBSBackend

class openff.evaluator.backends.dask.**DaskPBSBackend**(*minimum_number_of_workers=1, maximum_number_of_workers=1, resources_per_worker=<openff.evaluator.backends.backends.Queue object>, queue_name='default', setup_script_commands=None, extra_script_options=None, adaptive_interval='10000ms', disable_nanny_process=False, resource_line=None, adaptive_class=None*)

An openff-evaluator backend which uses a *dask_jobqueue.PBSCluster* object to run calculations within an existing PBS queue.

See also:

dask_jobqueue.LSFCluster, *DaskLSFBackend*

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

Constructs a new DaskLSFBackend object

Parameters **resource_line** (str) – The string to pass to the *#PBS -l* line.

Examples

To create a PBS 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 openff.evaluator.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',
>>> ]
>>>
>>> # 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 openff.evaluator.backends.dask import DaskPBSBackend
>>>
>>> pbs_backend = DaskPBSBackend(minimum_number_of_workers=1,
>>>                              maximum_number_of_workers=10,
```

(continues on next page)

(continued from previous page)

```
>>>                                 resources_per_worker=resources,
>>>                                 queue_name='gpuqueue',
>>>                                 setup_script_commands=setup_script_commands)
```

Methods

`__init__`([minimum_number_of_workers, ...])	Constructs a new DaskLSFBackend object
`job_script`()	Returns the job script that dask will use to submit workers.
`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.

Attributes

`started`	Returns whether this backend has been started yet.

`job_script()`
Returns the job script that dask will use to submit workers. The backend must be started before calling this function.

Returns

Return type str

`start()`
Start the calculation backend.

`property started`
Returns whether this backend has been started yet.

Type bool

`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

2.32.8 Storage API

StorageBackend	An abstract base representation of how the openff-evaluator will interact with and store simulation data.

StorageBackend

class `openff.evaluator.storage.`**`StorageBackend`**

An abstract base representation of how the openff-evaluator will interact with and store simulation data.

Notes

When implementing this class, only private methods should be overridden as the public methods only mainly implement thread locks, while their private version perform their actual function.

`__init__`()

Constructs a new StorageBackend object.

Methods

`__init__`()	Constructs a new StorageBackend object.
`has_force_field`(force_field)	A convenience method for checking whether the specified *ForceFieldSource* object is stored in the backend.
`has_object`(storage_object)	Checks whether a given hashable object exists in the storage system.
`query`(data_query)	Query the storage backend for data matching the query criteria.
`retrieve_force_field`(storage_key)	A convenience method for retrieving *ForceFieldSource* objects.
`retrieve_object`(storage_key[, expected_type])	Retrieves a stored object for the estimators storage system.
`store_force_field`(force_field)	A convenience method for storing *ForceFieldSource* objects.
`store_object`(object_to_store[, ...])	Store an object in the storage system, returning the key of the stored object.

`store_object`(*object_to_store*, *ancillary_data_path=None*)

Store an object in the storage system, returning the key of the stored object. This may be different to *storage_key* depending on whether the same or a similar object was already present in the system.

Parameters

- **`object_to_store`** (`BaseStoredData`) – The object to store.
- **`ancillary_data_path`** (*`str, optional`*) – The data path to the ancillary directory-like data to store alongside the object if the data type requires one.

Returns The unique key assigned to the stored object.

Return type str

store_force_field(*force_field*)
A convenience method for storing *ForceFieldSource* objects.

Parameters **force_field** (`ForceFieldSource`) – The force field to store.

Returns The unique id of the stored force field.

Return type str

retrieve_object(*storage_key*, *expected_type=None*)
Retrieves a stored object for the estimators storage system.

Parameters

- **storage_key** (*str*) – A unique key that describes where the stored object can be found within the storage system.
- **expected_type** (*type of BaseStoredData, optional*) – The expected data type. An exception is raised if the retrieved data doesn't match the type.

Returns

- *BaseStoredData, optional* – The stored object if the object key is found, otherwise None.
- *str, optional* – The path to the ancillary data if present.

retrieve_force_field(*storage_key*)
A convenience method for retrieving *ForceFieldSource* objects.

Parameters **storage_key** (*str*) – The key of the force field to retrieve.

Returns The retrieved force field source.

Return type *ForceFieldSource*

has_object(*storage_object*)
Checks whether a given hashable object exists in the storage system.

Parameters **storage_object** (`BaseStoredData`) – The object to check for.

Returns The unique key of the object if it is in the system, *None* otherwise.

Return type str, optional

has_force_field(*force_field*)
A convenience method for checking whether the specified *ForceFieldSource* object is stored in the backend.

Parameters **force_field** (`ForceFieldSource`) – The force field to look for.

Returns The unique key of the object if it is in the system, *None* otherwise.

Return type str, optional

query(*data_query*)
Query the storage backend for data matching the query criteria.

Parameters **data_query** (`BaseDataQuery`) – The query to perform.

Returns The data that matches the query partitioned by the matched values. The list values take the form (storage_key, data_object, data_directory_path).

Return type dict of tuple and list of tuple of str, BaseStoredData and str

Built-in Storage Backends

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

LocalFileStorage

class `openff.evaluator.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 StorageBackend object.

Methods

`__init__`([root_directory])	Constructs a new StorageBackend object.
`has_force_field`(force_field)	A convenience method for checking whether the specified *ForceFieldSource* object is stored in the backend.
`has_object`(storage_object)	Checks whether a given hashable object exists in the storage system.
`query`(data_query)	Query the storage backend for data matching the query criteria.
`retrieve_force_field`(storage_key)	A convenience method for retrieving *ForceField-Source* objects.
`retrieve_object`(storage_key[, expected_type])	Retrieves a stored object for the estimators storage system.
`store_force_field`(force_field)	A convenience method for storing *ForceFieldSource* objects.
`store_object`(object_to_store[, ...])	Store an object in the storage system, returning the key of the stored object.

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

`has_force_field`(*force_field*)

A convenience method for checking whether the specified *ForceFieldSource* object is stored in the backend.

Parameters **`force_field`** (`ForceFieldSource`) – The force field to look for.

Returns The unique key of the object if it is in the system, *None* otherwise.

Return type str, optional

`has_object`(*storage_object*)

Checks whether a given hashable object exists in the storage system.

Parameters **storage_object** (`BaseStoredData`) – The object to check for.

Returns The unique key of the object if it is in the system, *None* otherwise.

Return type str, optional

query(*data_query*)
Query the storage backend for data matching the query criteria.

Parameters **data_query** (`BaseDataQuery`) – The query to perform.

Returns The data that matches the query partitioned by the matched values. The list values take the form (storage_key, data_object, data_directory_path).

Return type dict of tuple and list of tuple of str, BaseStoredData and str

retrieve_force_field(*storage_key*)
A convenience method for retrieving *ForceFieldSource* objects.

Parameters **storage_key** (*str*) – The key of the force field to retrieve.

Returns The retrieved force field source.

Return type *ForceFieldSource*

retrieve_object(*storage_key*, *expected_type=None*)
Retrieves a stored object for the estimators storage system.

Parameters

- **storage_key** (*str*) – A unique key that describes where the stored object can be found within the storage system.
- **expected_type** (*type of BaseStoredData, optional*) – The expected data type. An exception is raised if the retrieved data doesn't match the type.

Returns

- *BaseStoredData, optional* – The stored object if the object key is found, otherwise None.
- *str, optional* – The path to the ancillary data if present.

store_force_field(*force_field*)
A convenience method for storing *ForceFieldSource* objects.

Parameters **force_field** (`ForceFieldSource`) – The force field to store.

Returns The unique id of the stored force field.

Return type str

store_object(*object_to_store*, *ancillary_data_path=None*)
Store an object in the storage system, returning the key of the stored object. This may be different to *storage_key* depending on whether the same or a similar object was already present in the system.

Parameters

- **object_to_store** (`BaseStoredData`) – The object to store.
- **ancillary_data_path** (*str, optional*) – The data path to the ancillary directory-like data to store alongside the object if the data type requires one.

Returns The unique key assigned to the stored object.

Return type str

Data Classes

`BaseStoredData`	A base representation of cached data to be stored by a storage backend.
`HashableStoredData`	Represents a class of data objects which can be rapidly compared / indexed by their hash values.
`ForceFieldData`	A data container for force field objects which will be saved to disk.
`ReplaceableData`	Represents a piece of stored data which can be replaced in a *StorageBackend* by another piece of data of the same type.
`BaseSimulationData`	A base class for classes which will store the outputs of a molecular simulation
`StoredSimulationData`	A representation of data which has been cached from a single previous simulation.
`StoredFreeEnergyData`	A representation of data which has been cached from an free energy calculation which computed the free energy difference between a start and end state.

BaseStoredData

class `openff.evaluator.storage.data.`**`BaseStoredData`**

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

The expectation is that stored data may 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 as a file name.

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 their file name by the data object.

The ancillary directory-like structure is not required if the data may be suitably stored in the data object itself.

`__init__`()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`has_ancillary_data`()	Returns whether this data object requires an accompanying data directory-like structure.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`to_storage_query`()	Returns the storage query which would match this data object.
`validate`([attribute_type])	Validate the values of the attributes.

abstract classmethod has_ancillary_data()

Returns whether this data object requires an accompanying data directory-like structure.

Returns True if this class requires an accompanying data directory-like structure.

Return type bool

to_storage_query()

Returns the storage query which would match this data object.

Returns The storage query which would match this data object.

Return type *BaseDataQuery*

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

HashableStoredData

class openff.evaluator.storage.data.**HashableStoredData**

Represents a class of data objects which can be rapidly compared / indexed by their hash values.

__init__()

Methods

__init__()	
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
has_ancillary_data()	Returns whether this data object requires an accompanying data directory-like structure.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
to_storage_query()	Returns the storage query which would match this data object.
validate([attribute_type])	Validate the values of the attributes.

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

abstract classmethod has_ancillary_data()

Returns whether this data object requires an accompanying data directory-like structure.

Returns True if this class requires an accompanying data directory-like structure.

Return type bool

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (`str or bytes`) – The typed json string.

Returns The parsed class.

Return type Any

to_storage_query()

Returns the storage query which would match this data object.

Returns The storage query which would match this data object.

Return type *BaseDataQuery*

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (`type of Attribute, optional`) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

ForceFieldData

class openff.evaluator.storage.data.**ForceFieldData**

A data container for force field objects which will be saved to disk.

__init__()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`has_ancillary_data`()	Returns whether this data object requires an accompanying data directory-like structure.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`to_storage_query`()	**returns** The storage query which would match this
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

force_field_source	The force field source object.

force_field_source
The force field source object. The default value of this attribute is not set and must be set by the user..

Type *ForceFieldSource*

classmethod has_ancillary_data()
Returns whether this data object requires an accompanying data directory-like structure.

Returns True if this class requires an accompanying data directory-like structure.

Return type bool

to_storage_query()

Returns The storage query which would match this data object.

Return type *SimulationDataQuery*

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

ReplaceableData

class `openff.evaluator.storage.data.`**ReplaceableData**

Represents a piece of stored data which can be replaced in a *StorageBackend* by another piece of data of the same type.

This may be the case for example when attempting to store a piece of *StoredSimulationData*, but another piece of data measured from the same calculation and for the same system already exists in the system, but stores less configurations.

__init__()

Methods

__init__()	
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
has_ancillary_data()	Returns whether this data object requires an accompanying data directory-like structure.
json([file_path, format])	Creates a JSON representation of this class.
most_information(stored_data_1, stored_data_2)	Returns the data object with the highest information content.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
to_storage_query()	Returns the storage query which would match this data object.
validate([attribute_type])	Validate the values of the attributes.

abstract classmethod most_information(*stored_data_1*, *stored_data_2*)

Returns the data object with the highest information content.

Parameters

- **stored_data_1** (`ReplaceableData`) – The first piece of data to compare.
- **stored_data_2** (`ReplaceableData`) – The second piece of data to compare.

Returns The data object with the highest information content, or *None* if the two pieces of information are incompatible with one another.

Return type *ReplaceableData*, optional

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

abstract classmethod has_ancillary_data()

Returns whether this data object requires an accompanying data directory-like structure.

Returns True if this class requires an accompanying data directory-like structure.

Return type bool

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

to_storage_query()

Returns the storage query which would match this data object.

Returns The storage query which would match this data object.

Return type *BaseDataQuery*

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

BaseSimulationData

class openff.evaluator.storage.data.**BaseSimulationData**

A base class for classes which will store the outputs of a molecular simulation

__init__()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`has_ancillary_data`()	Returns whether this data object requires an accompanying data directory-like structure.
`json`([file_path, format])	Creates a JSON representation of this class.
`most_information`(stored_data_1, stored_data_2)	Returns the data object with the highest information content.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`to_storage_query`()	Returns the storage query which would match this data object.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`force_field_id`	The id of the force field parameters used to generate the data.
`property_phase`	The phase of the system (e.g.
`source_calculation_id`	The server id of the calculation which yielded this data.
`substance`	A description of the composition of the stored system.
`thermodynamic_state`	The state at which the data was collected.

substance
A description of the composition of the stored system. The default value of this attribute is not set and must be set by the user..

Type *Substance*

thermodynamic_state
The state at which the data was collected. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

property_phase
The phase of the system (e.g. liquid, gas). The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

source_calculation_id
The server id of the calculation which yielded this data. The default value of this attribute is not set and must be set by the user..

Type str

force_field_id
The id of the force field parameters used to generate the data. The default value of this attribute is not set and must be set by the user..

Type str

`classmethod has_ancillary_data()`
Returns whether this data object requires an accompanying data directory-like structure.

Returns True if this class requires an accompanying data directory-like structure.

Return type bool

`classmethod from_json(`*file_path*`)`
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes(`*attribute_type=None*`)`
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json(`*file_path=None*, *format=False*`)`
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`abstract classmethod most_information(`*stored_data_1*, *stored_data_2*`)`
Returns the data object with the highest information content.

Parameters

- **`stored_data_1`** (`ReplaceableData`) – The first piece of data to compare.
- **`stored_data_2`** (`ReplaceableData`) – The second piece of data to compare.

Returns The data object with the highest information content, or *None* if the two pieces of information are incompatible with one another.

Return type *ReplaceableData*, optional

`classmethod parse_json(`*string_contents*`)`
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`to_storage_query()`
Returns the storage query which would match this data object.

Returns The storage query which would match this data object.

Return type *BaseDataQuery*

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

StoredSimulationData

class `openff.evaluator.storage.data.`**`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
```

`__init__`()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`has_ancillary_data`()	Returns whether this data object requires an accompanying data directory-like structure.
`json`([file_path, format])	Creates a JSON representation of this class.
`most_information`(stored_data_1, stored_data_2)	Returns the data object with the lowest *statistical_inefficiency*.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`to_storage_query`()	**returns** The storage query which would match this
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

coordinate_file_name	The name of a coordinate file which encodes the topology information of the system.
force_field_id	The id of the force field parameters used to generate the data.
number_of_molecules	The total number of molecules in the system.
observables	A frame of observables collected over the duration of the simulation.
property_phase	The phase of the system (e.g.
source_calculation_id	The server id of the calculation which yielded this data.
statistical_inefficiency	The statistical inefficiency of the collected data.
substance	A description of the composition of the stored system.
thermodynamic_state	The state at which the data was collected.
trajectory_file_name	The name of a .dcd trajectory file containing configurations generated by the simulation.

`coordinate_file_name`
The name of a coordinate file which encodes the topology information of the system. The default value of this attribute is not set and must be set by the user..

Type *FilePath*

`trajectory_file_name`
The name of a .dcd trajectory file containing configurations generated by the simulation. The default value of this attribute is not set and must be set by the user..

Type *FilePath*

`observables`
A frame of observables collected over the duration of the simulation. The default value of this attribute is not set and must be set by the user..

Type *ObservableFrame*

`statistical_inefficiency`
The statistical inefficiency of the collected data. The default value of this attribute is not set and must be set by the user..

Type float

`number_of_molecules`
The total number of molecules in the system. The default value of this attribute is not set and must be set by the user..

Type int

`classmethod most_information`(*stored_data_1*, *stored_data_2*)
Returns the data object with the lowest *statistical_inefficiency*.

Parameters

- `stored_data_1` (`StoredSimulationData`) – The first piece of data to compare.
- `stored_data_2` (`StoredSimulationData`) – The second piece of data to compare.

Returns

Return type *StoredSimulationData*

to_storage_query()

Returns The storage query which would match this data object.

Return type *SimulationDataQuery*

force_field_id

The id of the force field parameters used to generate the data. The default value of this attribute is not set and must be set by the user..

Type str

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (`str`) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (`type of Attribute, optional`) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

classmethod has_ancillary_data()

Returns whether this data object requires an accompanying data directory-like structure.

Returns True if this class requires an accompanying data directory-like structure.

Return type bool

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (`str, optional`) – The (optional) file path to save the JSON file to.
- **format** (`bool`) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (`str or bytes`) – The typed json string.

Returns The parsed class.

Return type Any

property_phase

The phase of the system (e.g. liquid, gas). The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

source_calculation_id
The server id of the calculation which yielded this data. The default value of this attribute is not set and must be set by the user..

Type str

substance
A description of the composition of the stored system. The default value of this attribute is not set and must be set by the user..

Type *Substance*

thermodynamic_state
The state at which the data was collected. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

StoredFreeEnergyData

class openff.evaluator.storage.data.**StoredFreeEnergyData**
A representation of data which has been cached from an free energy calculation which computed the free energy difference between a start and end state.

Notes

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

```
|--- data_object.json
|--- data_directory
     |--- topology_file_name.pdb
     |--- start_state_trajectory.dcd
     |--- end_state_trajectory.dcd
```

__init__()

Methods

__init__()	
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
has_ancillary_data()	Returns whether this data object requires an accompanying data directory-like structure.
json([file_path, format])	Creates a JSON representation of this class.

continues on next page

Table 206 – continued from previous page

`most_information`(stored_data_1, stored_data_2)	A comparison function which will always retain both pieces of free energy data.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`to_storage_query`()	**returns** The storage query which would match this data object.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`end_state_trajectory`	The name of a .dcd trajectory file containing configurations generated by the simulation of the end state of the system.
`force_field_id`	The id of the force field parameters used to generate the data.
`free_energy_difference`	The free energy difference between the end state and the start state.
`property_phase`	The phase of the system (e.g.
`source_calculation_id`	The server id of the calculation which yielded this data.
`start_state_trajectory`	The name of a .dcd trajectory file containing configurations generated by the simulation of the start state of the system.
`substance`	A description of the composition of the stored system.
`thermodynamic_state`	The state at which the data was collected.
`topology_file_name`	The name of a coordinate file which encodes the topology of the system.

`free_energy_difference`
The free energy difference between the end state and the start state. The default value of this attribute is not set and must be set by the user..

Type *Observable*

`topology_file_name`
The name of a coordinate file which encodes the topology of the system. The default value of this attribute is not set and must be set by the user..

Type *FilePath*

`start_state_trajectory`
The name of a .dcd trajectory file containing configurations generated by the simulation of the start state of the system. The default value of this attribute is not set and must be set by the user..

Type *FilePath*

`end_state_trajectory`
The name of a .dcd trajectory file containing configurations generated by the simulation of the end state of the system. The default value of this attribute is not set and must be set by the user..

Type *FilePath*

classmethod most_information(*stored_data_1:* openff.evaluator.storage.data.StoredFreeEnergyData, *stored_data_2:* openff.evaluator.storage.data.StoredFreeEnergyData) → Optional[*openff.evaluator.storage.data.StoredFreeEnergyData*]

A comparison function which will always retain both pieces of free energy data. At this time no situation can be envisaged that the same free energy data from exactly the same calculation will be store.

Parameters

- **stored_data_1** – The first piece of data to compare.
- **stored_data_2** – The second piece of data to compare.

to_storage_query()

Returns The storage query which would match this data object.

Return type *FreeEnergyDataQuery*

force_field_id

The id of the force field parameters used to generate the data. The default value of this attribute is not set and must be set by the user..

Type str

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

classmethod has_ancillary_data()

Returns whether this data object requires an accompanying data directory-like structure.

Returns True if this class requires an accompanying data directory-like structure.

Return type bool

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`property_phase`
The phase of the system (e.g. liquid, gas). The default value of this attribute is not set and must be set by the user..

Type *PropertyPhase*

`source_calculation_id`
The server id of the calculation which yielded this data. The default value of this attribute is not set and must be set by the user..

Type str

`substance`
A description of the composition of the stored system. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`thermodynamic_state`
The state at which the data was collected. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

Data Queries

`BaseDataQuery`	A base class for queries which can be made to a *StorageBackend*.
`SubstanceQuery`	A query which focuses on finding data which was collected for substances with specific traits, e.g which contains both a solute and solvent, or only a solvent etc.
`ForceFieldQuery`	A class used to query a *StorageBackend* for *ForceFieldData* which meet the specified criteria.
`BaseSimulationDataQuery`	The base class for queries which will retrieve `BaseSimulationData` derived data.
`SimulationDataQuery`	A class used to query a `StorageBackend` for `StoredSimulationData` objects which meet the specified set of criteria.
`FreeEnergyDataQuery`	A class used to query a `StorageBackend` for `FreeEnergyData` objects which meet the specified set of criteria.

BaseDataQuery

class openff.evaluator.storage.query.**BaseDataQuery**

A base class for queries which can be made to a *StorageBackend*.

__init__()

Methods

__init__()	
apply(data_object)	Apply this query to a data object.
data_class()	The type of data class that this query can be applied to.
from_data_object(data_object)	Returns the query which would match this data object.
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
validate([attribute_type])	Validate the values of the attributes.

abstract classmethod data_class()

The type of data class that this query can be applied to.

Returns

Return type type of BaseStoredData

apply(*data_object*)

Apply this query to a data object.

Parameters **data_object** (BaseStoredData) – The data object to apply the query to.

Returns The values of the matched parameters of the data object fully matched this query, otherwise *None*.

Return type tuple of Any, optional

classmethod from_data_object(*data_object*)

Returns the query which would match this data object.

Parameters **data_object** (BaseStoredData) – The data object to construct the query for.

Returns The query which would match this data object.

Return type cls

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

SubstanceQuery

`class openff.evaluator.storage.query.SubstanceQuery`

A query which focuses on finding data which was collected for substances with specific traits, e.g which contains both a solute and solvent, or only a solvent etc.

`__init__`()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

components_only	Only match pure data which was collected for one of the components in the query substance.

components_only
Only match pure data which was collected for one of the components in the query substance. The default value of this attribute is `False`.

Type bool

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

ForceFieldQuery

class openff.evaluator.storage.query.**ForceFieldQuery**

A class used to query a *StorageBackend* for *ForceFieldData* which meet the specified criteria.

__init__()

Methods

__init__()	
apply(data_object)	Apply this query to a data object.
data_class()	The type of data class that this query can be applied to.
from_data_object(data_object)	Returns the query which would match this data object.
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
validate([attribute_type])	Validate the values of the attributes.

Attributes

force_field_source	The force field source to query for.

classmethod data_class()

The type of data class that this query can be applied to.

Returns

Return type type of BaseStoredData

force_field_source

The force field source to query for. The default value of this attribute is not set. This attribute is *optional*.

Type *ForceFieldSource*

apply(*data_object*)

Apply this query to a data object.

Parameters **data_object** (BaseStoredData) – The data object to apply the query to.

Returns The values of the matched parameters of the data object fully matched this query, otherwise *None*.

Return type tuple of Any, optional

classmethod from_data_object(*data_object*)

Returns the query which would match this data object.

Parameters **data_object** (BaseStoredData) – The data object to construct the query for.

Returns The query which would match this data object.

Return type cls

`classmethod from_json(`*file_path*`)`
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes(`*attribute_type=None*`)`
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json(`*file_path=None*, *format=False*`)`
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json(`*string_contents*`)`
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`validate(`*attribute_type=None*`)`
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

BaseSimulationDataQuery

`class openff.evaluator.storage.query.BaseSimulationDataQuery`
The base class for queries which will retrieve `BaseSimulationData` derived data.

`__init__()`

Methods

`__init__`()	
`apply`(data_object[, attributes_to_ignore])	Apply this query to a data object.
`data_class`()	The type of data class that this query can be applied to.
`from_data_object`(data_object)	Returns the query which would match this data object.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`force_field_id`	The id of the force field parameters which used to generate the data.
`property_phase`	The phase of the substance (e.g.
`source_calculation_id`	The server id which should have generated this data.
`substance`	The substance which the data should have been collected for.
`substance_query`	The subset of the *substance* to query for.
`thermodynamic_state`	The state at which the data should have been collected.

substance
The substance which the data should have been collected for. Data for a subset of this substance can be queried for by using the *substance_query* attribute The default value of this attribute is not set. This attribute is *optional*.

Type *Substance*

substance_query
The subset of the *substance* to query for. This option can only be used when the *substance* attribute is set. The default value of this attribute is not set. This attribute is *optional*.

Type *SubstanceQuery*

thermodynamic_state
The state at which the data should have been collected. The default value of this attribute is not set. This attribute is *optional*.

Type *ThermodynamicState*

property_phase
The phase of the substance (e.g. liquid, gas). The default value of this attribute is not set. This attribute is *optional*.

Type *PropertyPhase*

source_calculation_id

The server id which should have generated this data. The default value of this attribute is not set. This attribute is *optional*.

Type str

force_field_id

The id of the force field parameters which used to generate the data. The default value of this attribute is not set. This attribute is *optional*.

Type str

apply(*data_object*, *attributes_to_ignore=None*)

Apply this query to a data object.

Parameters **data_object** (BaseStoredData) – The data object to apply the query to.

Returns The values of the matched parameters of the data object fully matched this query, otherwise *None*.

Return type tuple of Any, optional

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

abstract classmethod data_class()

The type of data class that this query can be applied to.

Returns

Return type type of BaseStoredData

classmethod from_data_object(*data_object*)

Returns the query which would match this data object.

Parameters **data_object** (BaseStoredData) – The data object to construct the query for.

Returns The query which would match this data object.

Return type cls

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

SimulationDataQuery

class openff.evaluator.storage.query.**SimulationDataQuery**

A class used to query a `StorageBackend` for `StoredSimulationData` objects which meet the specified set of criteria.

__init__()

Methods

__init__()	
apply(data_object[, attributes_to_ignore])	Apply this query to a data object.
data_class()	The type of data class that this query can be applied to.
from_data_object(data_object)	Returns the query which would match this data object.
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
validate([attribute_type])	Validate the values of the attributes.

Attributes

force_field_id	The id of the force field parameters which used to generate the data.
number_of_molecules	The total number of molecules in the system.
property_phase	The phase of the substance (e.g.
source_calculation_id	The server id which should have generated this data.
substance	The substance which the data should have been collected for.
substance_query	The subset of the *substance* to query for.

continues on next page

Table 217 – continued from previous page

thermodynamic_state	The state at which the data should have been collected.

classmethod data_class()
The type of data class that this query can be applied to.

Returns

Return type type of BaseStoredData

number_of_molecules
The total number of molecules in the system. The default value of this attribute is not set. This attribute is *optional*.

Type int

apply(*data_object*, *attributes_to_ignore=None*)
Apply this query to a data object.

Parameters **data_object** (BaseStoredData) – The data object to apply the query to.

Returns The values of the matched parameters of the data object fully matched this query, otherwise *None*.

Return type tuple of Any, optional

force_field_id
The id of the force field parameters which used to generate the data. The default value of this attribute is not set. This attribute is *optional*.

Type str

classmethod from_data_object(*data_object*)
Returns the query which would match this data object.

Parameters **data_object** (BaseStoredData) – The data object to construct the query for.

Returns The query which would match this data object.

Return type cls

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json(string_contents)`
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`property_phase`
The phase of the substance (e.g. liquid, gas). The default value of this attribute is not set. This attribute is *optional*.

Type *PropertyPhase*

`source_calculation_id`
The server id which should have generated this data. The default value of this attribute is not set. This attribute is *optional*.

Type str

`substance`
The substance which the data should have been collected for. Data for a subset of this substance can be queried for by using the *substance_query* attribute The default value of this attribute is not set. This attribute is *optional*.

Type *Substance*

`substance_query`
The subset of the *substance* to query for. This option can only be used when the *substance* attribute is set. The default value of this attribute is not set. This attribute is *optional*.

Type *SubstanceQuery*

`thermodynamic_state`
The state at which the data should have been collected. The default value of this attribute is not set. This attribute is *optional*.

Type *ThermodynamicState*

`validate(attribute_type=None)`
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

FreeEnergyDataQuery

class openff.evaluator.storage.query.**FreeEnergyDataQuery**

A class used to query a `StorageBackend` for `FreeEnergyData` objects which meet the specified set of criteria.

`__init__`()

Methods

`__init__`()	
`apply`(data_object[, attributes_to_ignore])	Apply this query to a data object.
`data_class`()	The type of data class that this query can be applied to.
`from_data_object`(data_object)	Returns the query which would match this data object.
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`force_field_id`	The id of the force field parameters which used to generate the data.
`property_phase`	The phase of the substance (e.g.
`source_calculation_id`	The server id which should have generated this data.
`substance`	The substance which the data should have been collected for.
`substance_query`	The subset of the *substance* to query for.
`thermodynamic_state`	The state at which the data should have been collected.

classmethod data_class()

The type of data class that this query can be applied to.

Returns

Return type type of BaseStoredData

apply(*data_object*, *attributes_to_ignore=None*)

Apply this query to a data object.

Parameters **data_object** (`BaseStoredData`) – The data object to apply the query to.

Returns The values of the matched parameters of the data object fully matched this query, otherwise *None*.

Return type tuple of Any, optional

force_field_id

The id of the force field parameters which used to generate the data. The default value of this attribute is not set. This attribute is *optional*.

Type str

classmethod from_data_object(*data_object*)

Returns the query which would match this data object.

Parameters **data_object** (BaseStoredData) – The data object to construct the query for.

Returns The query which would match this data object.

Return type cls

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

property_phase

The phase of the substance (e.g. liquid, gas). The default value of this attribute is not set. This attribute is *optional*.

Type *PropertyPhase*

source_calculation_id

The server id which should have generated this data. The default value of this attribute is not set. This attribute is *optional*.

Type str

substance
The substance which the data should have been collected for. Data for a subset of this substance can be queried for by using the *substance_query* attribute The default value of this attribute is not set. This attribute is *optional*.

Type *Substance*

substance_query
The subset of the *substance* to query for. This option can only be used when the *substance* attribute is set. The default value of this attribute is not set. This attribute is *optional*.

Type *SubstanceQuery*

thermodynamic_state
The state at which the data should have been collected. The default value of this attribute is not set. This attribute is *optional*.

Type *ThermodynamicState*

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

Attributes

FilePath	Represents a string file path.
StorageAttribute	A descriptor used to mark attributes of a class as those which store information about a cached piece of data.
QueryAttribute	A descriptor used to add additional metadata to attributes of a storage query.

FilePath

class openff.evaluator.storage.attributes.**FilePath**
Represents a string file path.

__init__()

Methods

__init__()	
capitalize()	Return a capitalized version of the string.
casefold()	Return a version of the string suitable for caseless comparisons.
center(width[, fillchar])	Return a centered string of length width.
count(sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
encode([encoding, errors])	Encode the string using the codec registered for encoding.

continues on next page

Table 221 – continued from previous page

endswith(suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
expandtabs([tabsize])	Return a copy where all tab characters are expanded using spaces.
find(sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
format(*args, **kwargs)	Return a formatted version of S, using substitutions from args and kwargs.
format_map(mapping)	Return a formatted version of S, using substitutions from mapping.
index(sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
isalnum()	Return True if the string is an alpha-numeric string, False otherwise.
isalpha()	Return True if the string is an alphabetic string, False otherwise.
isascii()	Return True if all characters in the string are ASCII, False otherwise.
isdecimal()	Return True if the string is a decimal string, False otherwise.
isdigit()	Return True if the string is a digit string, False otherwise.
isidentifier()	Return True if the string is a valid Python identifier, False otherwise.
islower()	Return True if the string is a lowercase string, False otherwise.
isnumeric()	Return True if the string is a numeric string, False otherwise.
isprintable()	Return True if the string is printable, False otherwise.
isspace()	Return True if the string is a whitespace string, False otherwise.
istitle()	Return True if the string is a title-cased string, False otherwise.
isupper()	Return True if the string is an uppercase string, False otherwise.
join(iterable, /)	Concatenate any number of strings.
ljust(width[, fillchar])	Return a left-justified string of length width.
lower()	Return a copy of the string converted to lowercase.
lstrip([chars])	Return a copy of the string with leading whitespace removed.
maketrans(x[, y, z])	Return a translation table usable for str.translate().
partition(sep, /)	Partition the string into three parts using the given separator.
replace(old, new[, count])	Return a copy with all occurrences of substring old replaced by new.
rfind(sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
rindex(sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
rjust(width[, fillchar])	Return a right-justified string of length width.

continues on next page

Table 221 – continued from previous page

`rpartition`(sep, /)	Partition the string into three parts using the given separator.
`rsplit`([sep, maxsplit])	Return a list of the words in the string, using sep as the delimiter string.
`rstrip`([chars])	Return a copy of the string with trailing whitespace removed.
`split`([sep, maxsplit])	Return a list of the words in the string, using sep as the delimiter string.
`splitlines`([keepends])	Return a list of the lines in the string, breaking at line boundaries.
`startswith`(prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
`strip`([chars])	Return a copy of the string with leading and trailing whitespace removed.
`swapcase`()	Convert uppercase characters to lowercase and lowercase characters to uppercase.
`title`()	Return a version of the string where each word is titlecased.
`translate`(table, /)	Replace each character in the string using the given translation table.
`upper`()	Return a copy of the string converted to uppercase.
`zfill`(width, /)	Pad a numeric string with zeros on the left, to fill a field of the given width.

`capitalize()`
Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

`casefold()`
Return a version of the string suitable for caseless comparisons.

`center`(*width*, *fillchar=' '*, */*)
Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

`count`(*sub*[, *start*[, *end*]]) → int
Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

`encode`(*encoding='utf-8'*, *errors='strict'*)
Encode the string using the codec registered for encoding.

encoding The encoding in which to encode the string.

errors The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

`endswith`(*suffix*[, *start*[, *end*]]) → bool
Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

`expandtabs`(*tabsize=8*)
Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int
Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str
Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str
Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int
Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()
Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()
Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()
Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()
Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()
Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()
Return True if the string is a valid Python identifier, False otherwise.

Use keyword.iskeyword() to test for reserved identifiers such as "def" and "class".

islower()
Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()
Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

`isprintable()`
Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

`isspace()`
Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

`istitle()`
Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

`isupper()`
Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

`join`(*iterable*, /)
Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

`ljust`(*width*, *fillchar=' '*, /)
Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

`lower()`
Return a copy of the string converted to lowercase.

`lstrip`(*chars=None*, /)
Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

`static maketrans`(*x*, *y=None*, *z=None*, /)
Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

`partition`(*sep*, /)
Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

`replace`(*old*, *new*, *count=-1*, /)
Return a copy with all occurrences of substring old replaced by new.

count Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

`rfind`(*sub*[, *start*[, *end*]]) → int
Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

`rindex`(*sub*[, *start*[, *end*]]) → int
Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

`rjust`(*width*, *fillchar=' '*, */*)
Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

`rpartition`(*sep*, */*)
Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

`rsplit`(*sep=None*, *maxsplit=- 1*)
Return a list of the words in the string, using sep as the delimiter string.

sep The delimiter according which to split the string. None (the default value) means split according to any whitespace, and discard empty strings from the result.

maxsplit Maximum number of splits to do. -1 (the default value) means no limit.

Splits are done starting at the end of the string and working to the front.

`rstrip`(*chars=None*, */*)
Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

`split`(*sep=None*, *maxsplit=- 1*)
Return a list of the words in the string, using sep as the delimiter string.

sep The delimiter according which to split the string. None (the default value) means split according to any whitespace, and discard empty strings from the result.

maxsplit Maximum number of splits to do. -1 (the default value) means no limit.

`splitlines`(*keepends=False*)
Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

`startswith`(*prefix*[, *start*[, *end*]]) → bool
Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

`strip`(*chars=None*, */*)
Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()
Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()
Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table*, */*)
Replace each character in the string using the given translation table.

table Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via __getitem__, for instance a dictionary or list. If this operation raises LookupError, the character is left untouched. Characters mapped to None are deleted.

upper()
Return a copy of the string converted to uppercase.

zfill(*width*, */*)
Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

StorageAttribute

class openff.evaluator.storage.attributes.**StorageAttribute**(*docstring*, *type_hint*, *optional=False*)
A descriptor used to mark attributes of a class as those which store information about a cached piece of data.

__init__(*docstring*, *type_hint*, *optional=False*)
Initializes a new Attribute object.

Parameters

- **docstring** (*str*) – A docstring describing the attributes purpose. This will automatically be decorated with additional information such as type hints, default values, etc.
- **type_hint** (*type, typing.Union*) – The expected type of this attribute. This will be used to help the workflow engine ensure that expected input types match corresponding output values.
- **default_value** (*Any*) – The default value for this attribute.
- **optional** (*bool*) – Defines whether this is an optional input of a class. If true, the *default_value* should be set to *UNDEFINED*.
- **read_only** (*bool*) – Defines whether this attribute is read-only.

Methods

__init__(docstring, type_hint[, optional])	Initializes a new Attribute object.

QueryAttribute

class openff.evaluator.storage.attributes.**QueryAttribute**(*docstring*, *type_hint*, *optional=False*, *custom_match=False*)

A descriptor used to add additional metadata to attributes of a storage query.

__init__(*docstring*, *type_hint*, *optional=False*, *custom_match=False*)

Initializes self.

Parameters **custom_match** (*bool*) – Whether a custom behaviour will be implemented when matching this attribute against the matching data object attribute.

Methods

__init__(docstring, type_hint[, optional, ...])	Initializes self.

2.32.9 Workflow API

Workflow	Encapsulates and prepares a workflow which is able to estimate a physical property.
WorkflowException	An exception which was raised while executing a workflow protocol.
WorkflowGraph	A hierarchical structure for storing and submitting the workflows which will estimate a set of physical properties..
WorkflowResult	The result of executing a *Workflow* as part of a *WorkflowGraph*.
Protocol	The base class for a protocol which would form one step of a larger property calculation workflow.
ProtocolGraph	A graph of connected protocols which may be executed together.
ProtocolGroup	A group of workflow protocols to be executed in one batch.
workflow_protocol	A decorator which registers a class as being a protocol which may be included in workflows.
register_workflow_protocol	Registers a class as being a protocol which may be included in workflows.

Workflow

class openff.evaluator.workflow.**Workflow**(*global_metadata*, *unique_id=None*)

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

__init__(*global_metadata*, *unique_id=None*)

Constructs a new Workflow object.

Parameters

- **global_metadata** (*dict of str and Any*) – A dictionary of the metadata which will be made available to each of the workflow protocols through the pseudo "global" scope.
- **unique_id** (*str, optional*) – A unique identifier to assign to this workflow. This id will be appended to the ids of the protocols of this workflow. If none is provided, one will be chosen at random.

Methods

__init__(global_metadata[, unique_id])	Constructs a new Workflow object.
execute([root_directory, ...])	Executes the workflow.
from_schema(schema, metadata[, unique_id])	Creates a workflow from its schema blueprint, and the associated metadata.
generate_default_metadata(physical_property, ...)	Generates the default global metadata dictionary.
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.
to_graph()	Converts this workflow to an executable *WorkflowGraph*.

Attributes

final_value_source	The path to the protocol output which corresponds to the estimated value of the property being estimated.
outputs_to_store	A collection of data classes to populate ready to be stored by a *StorageBackend*.
protocols	The protocols in this workflow.
schema	

property protocols

The protocols in this workflow.

Type tuple of Protocol

property final_value_source

The path to the protocol output which corresponds to the estimated value of the property being estimated.

Type *ProtocolPath*

property outputs_to_store

A collection of data classes to populate ready to be stored by a *StorageBackend*.

Type dict of str and StorageBackend

replace_protocol(*old_protocol*, *new_protocol*, *update_paths_only=False*)

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** (`Protocol or ProtocolPath`) – The protocol (or its id) to replace.
- **new_protocol** (`Protocol or ProtocolPath`) – The new protocol (or its id) to use.
- **update_paths_only** (`bool`) – Whether only update the *final_value_source*, and *outputs_to_store* attributes, or to also update all of the protocols in *protocols*.

static generate_default_metadata(*physical_property*, *force_field_path*, *parameter_gradient_keys=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *target_uncertainty=None*)

Generates the 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.
- **target_uncertainty** (`openff.evaluator.unit.Quantity, optional`) – The uncertainty which the property should be estimated to within.

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: openff.evaluator.unit.Quantity - The target uncertainty with which** properties should be estimated.
- **per_component_uncertainty: openff.evaluator.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

to_graph()

Converts this workflow to an executable *WorkflowGraph*.

Returns The graph representation of this workflow.

Return type *WorkflowGraph*

`classmethod from_schema(schema, metadata, unique_id=None)`
Creates a workflow from its schema blueprint, and the associated metadata.

Parameters

- **schema** (`WorkflowSchema`) – The schema blueprint for this workflow.
- **metadata** (*`dict of str and Any`*) – The metadata to make available to the workflow.
- **unique_id** (*`str, optional`*) – A unique identifier to assign to this workflow. This id will be appended to the ids of the protocols of this workflow. If none is provided one will be chosen at random.

Returns The created workflow.

Return type cls

`execute(root_directory='', calculation_backend=None, compute_resources=None)`
Executes the workflow.

Parameters

- **root_directory** (*`str`*) – The directory to execute the graph in.
- **calculation_backend** (`CalculationBackend`, *`optional.`*) – The backend to execute the graph on. This parameter is mutually exclusive with *compute_resources*.
- **compute_resources** (`CalculationBackend`, *`optional.`*) – The compute resources to run using. If None and no *calculation_backend* is specified, the workflow will be executed on a single CPU thread. This parameter is mutually exclusive with *calculation_backend*.

Returns The result of executing this workflow. If executed on a *calculation_backend*, the result will be wrapped in a *Future* object.

Return type *WorkflowResult* or Future of WorkflowResult

WorkflowException

`exception openff.evaluator.workflow.WorkflowException(message=None, protocol_id=None)`
An exception which was raised while executing a workflow protocol.

`classmethod from_exception(exception)`
Initialize this class from an existing exception.

Parameters **exception** (*`Exception`*) – The existing exception

Returns The initialized exception object.

Return type cls

`classmethod from_json(file_path)`
Create this object from a JSON file.

Parameters **file_path** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`json(file_path=None, format=False)`
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

WorkflowGraph

class openff.evaluator.workflow.**WorkflowGraph**

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

__init__()

Methods

__init__()	
add_workflows(*workflows)	Insert a set of workflows into the workflow graph.
execute([root_directory, ...])	Executes the workflow graph.

Attributes

protocols	The protocols in this graph.
root_protocols	The ids of the protocols in the group which do not take input from the other grouped protocols.

property protocols

The protocols in this graph.

Type dict of str and Protocol

property root_protocols

The ids of the protocols in the group which do not take input from the other grouped protocols.

Type list of str

add_workflows(**workflows*)

Insert a set of workflows into the workflow graph.

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

execute(*root_directory=''*, *calculation_backend=None*, *compute_resources=None*)
Executes the workflow graph.

Parameters

- **root_directory** (`str`) – The directory to execute the graph in.
- **calculation_backend** (`CalculationBackend, optional.`) – The backend to execute the graph on. This parameter is mutually exclusive with *compute_resources*.
- **compute_resources** (`CalculationBackend, optional.`) – The compute resources to run using. If None and no *calculation_backend* is specified, the workflow will be executed on a single CPU thread. This parameter is mutually exclusive with *calculation_backend*.

Returns The results of executing the graph. If a *calculation_backend* is specified, these results will be wrapped in a *Future*.

Return type list of WorkflowResult or list of Future of WorkflowResult

WorkflowResult

class openff.evaluator.workflow.**WorkflowResult**
The result of executing a *Workflow* as part of a *WorkflowGraph*.

__init__()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`data_to_store`	Paths to the data objects to store.
`exceptions`	Any exceptions raised by the layer while estimating the property.
`gradients`	The gradients of the estimated value with respect to the specified force field parameters.
`value`	The estimated value of the property and the uncertainty in that value.
`workflow_id`	The id of the workflow associated with this result.

workflow_id
The id of the workflow associated with this result. The default value of this attribute is not set and must be set by the user..

Type str

`value`
The estimated value of the property and the uncertainty in that value. The default value of this attribute is not set. This attribute is *optional*.

Type Measurement

`gradients`
The gradients of the estimated value with respect to the specified force field parameters. The default value of this attribute is `[]`.

Type list

`exceptions`
Any exceptions raised by the layer while estimating the property. The default value of this attribute is `[]`.

Type list

`data_to_store`
Paths to the data objects to store. The default value of this attribute is `[]`.

Type list

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

Protocol

class openff.evaluator.workflow.**Protocol**(*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 openff.evaluator.protocols.openmm import OpenMMSimulation
>>>
>>> npt_equilibration = OpenMMSimulation('npt_equilibration')
>>> npt_equilibration.ensemble = OpenMMSimulation.Ensemble.NPT
```

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

```
>>> npt_production = OpenMMSimulation('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.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

continues on next page

Table 231 – 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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

Type bool

property schema

A serializable schema for this object.

Type *ProtocolSchema*

property required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

property outputs

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

property dependencies

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

Type list of ProtocolPath

id

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

classmethod from_schema(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (*Protocol*) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

merge(*other*)

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

Parameters **other** (Protocol) – 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`** (`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_class_attribute`(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

ProtocolGraph

class openff.evaluator.workflow.**ProtocolGraph**

A graph of connected protocols which may be executed together.

__init__()

Methods

__init__()	
add_protocols(*protocols[, ...])	Adds a set of protocols to the graph.
execute([root_directory, ...])	Execute the protocol graph in the specified directory, and either using a *CalculationBackend*, or using a specified set of compute resources.

Attributes

protocols	The protocols in this graph.
root_protocols	The ids of the protocols in the group which do not take input from the other grouped protocols.

property protocols

The protocols in this graph.

Type dict of str and Protocol

property root_protocols

The ids of the protocols in the group which do not take input from the other grouped protocols.

Type list of str

add_protocols(**protocols*, *allow_external_dependencies=False*)

Adds a set of protocols to the graph.

Parameters

- **protocols** (*tuple of Protocol*) – The protocols to add.
- **allow_external_dependencies** (*bool*) – If *False*, an exception will be raised if a protocol has a dependency outside of this graph.

Returns A mapping between the original protocols and protocols which were merged over the course of adding the new protocols.

Return type dict of str and str

execute(*root_directory=''*, *calculation_backend=None*, *compute_resources=None*, *enable_checkpointing=True*, *safe_exceptions=True*)

Execute the protocol graph in the specified directory, and either using a *CalculationBackend*, or using a specified set of compute resources.

Parameters

- **root_directory** (*str*) – The directory to execute the graph in.
- **calculation_backend** (CalculationBackend, *optional.*) – The backend to execute the graph on. This parameter is mutually exclusive with *compute_resources*.
- **compute_resources** (CalculationBackend, *optional.*) – The compute resources to run using. This parameter is mutually exclusive with *calculation_backend*.
- **enable_checkpointing** (*bool*) – If enabled, protocols will not be executed more than once if the output from their previous execution is found.
- **safe_exceptions** (*bool*) – If true, exceptions will be serialized into the results file rather than directly raised, otherwise, the exception will be raised as normal.

Returns The paths to the JSON serialized outputs of the executed protocols. If executed using a calculation backend, these will be *Future* objects which will return the output paths on calling *future.result()*.

Return type dict of str and str or Future

ProtocolGroup

class openff.evaluator.workflow.**ProtocolGroup**(*protocol_id*)

A group of workflow 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)	Add protocols to this group.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.

continues on next page

Table 235 – continued from previous page

merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
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.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
protocols	A dictionary of the protocols in this groups, where the dictionary key is the protocol id, and the value is the protocol itself.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.

property required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

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

Type list of ProtocolPath

property outputs
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

property protocols
A dictionary of the protocols in this groups, where the dictionary key is the protocol id, and the value is the protocol itself.

Notes

This property should *not* be altered. Use *add_protocols* to add new protocols to the group.

Type dict of str and Protocol

add_protocols(**protocols*)
Add protocols to this group.

Parameters protocols (Protocol) – The protocols 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.

can_merge(*other*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (*Protocol*) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

merge(*other*)

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

Parameters **other** (Protocol) – 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** (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_class_attribute(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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

`allow_merging`
Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

Type bool

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **`schema`** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

property schema
A serializable schema for this object.

Type *ProtocolSchema*

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

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

workflow_protocol

`openff.evaluator.workflow.`**`workflow_protocol`**`()`

A decorator which registers a class as being a protocol which may be included in workflows.

register_workflow_protocol

`openff.evaluator.workflow.`**`register_workflow_protocol`**`(`*protocol_class*`)`

Registers a class as being a protocol which may be included in workflows.

Schemas

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.
WorkflowSchema	The schematic for a property estimation workflow.

ProtocolSchema

class `openff.evaluator.workflow.schemas.`**`ProtocolSchema`**`(`*unique_id=None*, *protocol_type=None*, *inputs=None*`)`

A json serializable representation of a workflow protocol.

`__init__``(`*unique_id=None*, *protocol_type=None*, *inputs=None*`)`

Methods

__init__([unique_id, protocol_type, inputs])	
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
to_protocol()	Creates a new protocol object from this schema.
validate([attribute_type])	Validate the values of the attributes.

Attributes

id	The unique id associated with the protocol.
inputs	The inputs to the protocol.
type	The type of protocol associated with this schema.

id

The unique id associated with the protocol. The default value of this attribute is not set and must be set by the user..

Type str

type

The type of protocol associated with this schema. The default value of this attribute is not set and must be set by the user.. This attribute is *read-only*.

Type str

inputs

The inputs to the protocol. The default value of this attribute is not set and must be set by the user.. This attribute is *read-only*.

Type dict

to_protocol()

Creates a new protocol object from this schema.

Returns The protocol created from this schema.

Return type *Protocol*

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

ProtocolGroupSchema

class openff.evaluator.workflow.schemas.**ProtocolGroupSchema**(*unique_id=None*, *protocol_type=None*, *inputs=None*, *protocol_schemas=None*)

A json serializable representation of a workflow protocol group.

__init__(*unique_id=None*, *protocol_type=None*, *inputs=None*, *protocol_schemas=None*)

Methods

__init__([unique_id, protocol_type, inputs, ...])	

from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
to_protocol()	Creates a new protocol object from this schema.
validate([attribute_type])	Validate the values of the attributes.

Attributes

id	The unique id associated with the protocol.
inputs	The inputs to the protocol.
protocol_schemas	The schemas of the protocols within this group.
type	The type of protocol associated with this schema.

protocol_schemas

The schemas of the protocols within this group. The default value of this attribute is not set and must be set by the user.. This attribute is *read-only*.

Type dict

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`classmethod` **`from_json`**(*file_path*)

Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod` **`get_attributes`**(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`id`

The unique id associated with the protocol. The default value of this attribute is not set and must be set by the user..

Type str

`inputs`

The inputs to the protocol. The default value of this attribute is not set and must be set by the user.. This attribute is *read-only*.

Type dict

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`classmethod` **`parse_json`**(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`to_protocol`()

Creates a new protocol object from this schema.

Returns The protocol created from this schema.

Return type *Protocol*

type
The type of protocol associated with this schema. The default value of this attribute is not set and must be set by the user.. This attribute is *read-only*.

Type str

ProtocolReplicator

class `openff.evaluator.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.
`from_json`(file_path)	Create this object from a JSON file.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	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 Protocol*) – 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 Protocol* – 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 Protocol*) – 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.

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

WorkflowSchema

class openff.evaluator.workflow.schemas.**WorkflowSchema**

The schematic for a property estimation workflow.

__init__()

Methods

__init__()	
from_json(file_path)	Create this object from a JSON file.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
json([file_path, format])	Creates a JSON representation of this class.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol_types(protocol_replacements)	Replaces protocols with given types with other protocols of specified replacements.
validate([attribute_type])	Validate the values of the attributes.

Attributes

final_value_source	A reference to which protocol output corresponds to the estimated value of the property.
outputs_to_store	A collection of data classes to populate ready to be stored by a *StorageBackend*.
protocol_replicators	A set of replicators which will replicate parts of the workflow.
protocol_schemas	The schemas for the protocols which will make up the workflow.

protocol_schemas
The schemas for the protocols which will make up the workflow. The default value of this attribute is `[]`.

Type list

protocol_replicators
A set of replicators which will replicate parts of the workflow. The default value of this attribute is not set. This attribute is *optional*.

Type list

final_value_source
A reference to which protocol output corresponds to the estimated value of the property. The default value of this attribute is not set. This attribute is *optional*.

Type *ProtocolPath*

outputs_to_store
A collection of data classes to populate ready to be stored by a *StorageBackend*. The default value of this attribute is not set. This attribute is *optional*.

Type dict

replace_protocol_types(*protocol_replacements*, *protocol_group_schema=None*)
Replaces protocols with given types with other protocols of specified replacements. This is useful when replacing the default protocols with custom ones, or swapping out base protocols with actual implementations

Warning: This method is NOT fully implemented and is likely to fail in all but a few specific cases. This method should be used with extreme caution.

Parameters

- **protocol_replacements** (*dict of str and str, optional*) – A dictionary with keys of the types of protocols which should be replaced with those protocols named by the values.
- **protocol_group_schema** (`ProtocolGroupSchema`) – The protocol group to apply the replacements to. This is mainly used when applying this method recursively.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

classmethod **`from_json`**(*file_path*)

Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod **`get_attributes`**(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod **`parse_json`**(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

Attributes

`BaseMergeBehaviour`	A base class for enums which will describes how attributes should be handled when attempting to merge similar protocols.
`MergeBehaviour`	A enum which describes how attributes should be handled when attempting to merge similar protocols.
`InequalityMergeBehaviour`	A enum which describes how attributes which can be compared with inequalities should be merged.
`InputAttribute`	A descriptor used to mark an attribute of an object as an input to that object.
`OutputAttribute`	A descriptor used to mark an attribute of an as an output of that object.

BaseMergeBehaviour

class `openff.evaluator.workflow.attributes.`**`BaseMergeBehaviour`**`(`*value*`)`

A base class for enums which will describes how attributes should be handled when attempting to merge similar protocols.

`__init__``()`

MergeBehaviour

class `openff.evaluator.workflow.attributes.`**`MergeBehaviour`**`(`*value*`)`

A enum which describes how attributes should be handled when attempting to merge similar protocols.

This enum may take values of

- ExactlyEqual: This attribute must be exactly equal between two protocols for them to be able to merge.
- Custom: This attribute will be ignored by the built-in merging code such that user specified behavior can be implemented.

`__init__``()`

Attributes

`ExactlyEqual`
`Custom`

InequalityMergeBehaviour

class `openff.evaluator.workflow.attributes.`**`InequalityMergeBehaviour`**`(`*value*`)`

A enum which describes how attributes which can be compared with inequalities should be merged.

This enum may take values of

- SmallestValue: When two protocols are merged, the smallest value of this attribute from either protocol is retained.
- LargestValue: When two protocols are merged, the largest value of this attribute from either protocol is retained.

`__init__``()`

Attributes

`SmallestValue`
`LargestValue`

InputAttribute

class openff.evaluator.workflow.attributes.**InputAttribute**(*docstring*, *type_hint*, *default_value*, *optional=False*, *merge_behavior=MergeBehaviour.ExactlyEqual*)

A descriptor used to mark an attribute of an object as an input to that object.

An attribute can either be set with a value directly, or it can also be set to a *ProtocolPath* to be set be the workflow manager.

Examples

To mark an attribute as an input:

```
>>> from openff.evaluator.attributes import AttributeClass
>>> from openff.evaluator.workflow.attributes import InputAttribute
>>>
>>> class MyObject(AttributeClass):
>>>
>>>     my_input = InputAttribute(
>>>         docstring='An input will be used.',
>>>         type_hint=float,
>>>         default_value=0.1
>>>     )
```

__init__(*docstring*, *type_hint*, *default_value*, *optional=False*, *merge_behavior=MergeBehaviour.ExactlyEqual*)

Initializes a new InputAttribute object.

Parameters **merge_behavior** (BaseMergeBehaviour) – An enum describing how this input should be handled when considering whether to, and actually merging two different objects.

Methods

__init__(docstring, type_hint, default_value)	Initializes a new InputAttribute object.

OutputAttribute

class openff.evaluator.workflow.attributes.**OutputAttribute**(*docstring*, *type_hint*)

A descriptor used to mark an attribute of an as an output of that object. This attribute is expected to be populated by the object itself, rather than be set externally.

Examples

To mark an attribute as an output:

```
>>> from openff.evaluator.attributes import AttributeClass
>>> from openff.evaluator.workflow.attributes import OutputAttribute
>>>
>>> class MyObject(AttributeClass):
>>>
>>>     my_output = OutputAttribute(
>>>         docstring='An output that will be filled.',
>>>         type_hint=float
>>>     )
```

`__init__`(*docstring*, *type_hint*)
Initializes a new OutputAttribute object.

Methods

`__init__`(docstring, type_hint)	Initializes a new OutputAttribute object.

Placeholder Values

`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.

ReplicatorValue

`class openff.evaluator.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 openff.evaluator.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
copy()	Returns a copy of this 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.

Attributes

full_path	The full path referenced by this object.
is_global	
last_protocol	The end protocol id of the path.
path_separator	
property_name	The property name pointed to by the path.
property_separator	
protocol_ids	The ids of the protocols referenced by this object.
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 protocol_ids
The ids of the protocols referenced by this object.

Type tuple of str

property start_protocol

The leading protocol id of the path.

Type str

property last_protocol
The end 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

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.

copy()
Returns a copy of this path.

2.32.10 Built-in Workflow Protocols

Analysis

BaseAverageObservable	An abstract base class for protocols which will calculate the average value of an observable and its uncertainty via bootstrapping.
AverageObservable	Computes the average value of an observable as well as bootstrapped uncertainties for the average.

continues on next page

Table 255 – continued from previous page

AverageDielectricConstant	Computes the average value of the dielectric constant from a set of dipole moments (M) and volumes (V) sampled over the course of a molecular simulation such that `eps = 1 + (<M^2> - <M>^2) / (3.0 * eps_0 * <V> * kb * T)` [1]_.
AverageFreeEnergies	A protocol which computes the Boltzmann weighted average (G° = -RT × Log[_{n} exp(-G°_{n})]) of a set of free energies which were measured at the same thermodynamic state.
ComputeDipoleMoments	A protocol which will compute the dipole moment for each configuration in a trajectory and for a given parameterized system.
BaseDecorrelateProtocol	An abstract base class for protocols which will subsample a set of data, yielding only equilibrated, uncorrelated data.
DecorrelateTrajectory	A protocol which will subsample frames from a trajectory, yielding only uncorrelated frames as determined from a provided statistical inefficiency and equilibration time.
DecorrelateObservables	A protocol which will subsample a trajectory of observables, yielding only uncorrelated entries as determined from a provided statistical inefficiency and equilibration time.

BaseAverageObservable

class openff.evaluator.protocols.analysis.**BaseAverageObservable**(*protocol_id*)

An abstract base class for protocols which will calculate the average value of an observable and its uncertainty via bootstrapping.

`__init__`(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.

continues on next page

Table 256 – continued from previous page

json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
bootstrap_iterations	**Input** - The number of bootstrap iterations to perform.
bootstrap_sample_size	**Input** - The relative sample size to use for bootstrapping.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
potential_energies	**Input** - The potential energies which were evaluated at the same configurations and using the same force field parameters as the observable to average.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
thermodynamic_state	**Input** - The state at which the observables were computed.
time_series_statistics	**Output** - Statistics about the observables from which the average was computed.These include the statistical inefficiency and the index after which the observables have become stationary (i.e.
value	**Output** - The average value of the observable.

bootstrap_iterations

Input - The number of bootstrap iterations to perform. The default value of this attribute is `250`.

Type int

bootstrap_sample_size

Input - The relative sample size to use for bootstrapping. The default value of this attribute is `1.0`.

Type float

thermodynamic_state

Input - The state at which the observables were computed. This is required to compute ensemble averages of the gradients of the observable with respect to force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type *ThermodynamicState*

potential_energies

Input - The potential energies which were evaluated at the same configurations and using the same force field parameters as the observable to average. This is required to compute ensemble averages of the gradients of the observable with respect to force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type *ObservableArray*

`value`

Output - The average value of the observable. The default value of this attribute is not set and must be set by the user..

Type *Observable*

`time_series_statistics`

Output - Statistics about the observables from which the average was computed.These include the statistical inefficiency and the index after which the observables have become stationary (i.e. equilibrated). The default value of this attribute is not set and must be set by the user..

Type TimeSeriesStatistics

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`

The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (`str`) – The uuid to prepend.

`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.

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (`type of Attribute, optional`) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

AverageObservable

`class openff.evaluator.protocols.analysis.`**`AverageObservable`**(*protocol_id*)

Computes the average value of an observable as well as bootstrapped uncertainties for the average.

`__init__`(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`bootstrap_iterations`	**Input** - The number of bootstrap iterations to perform.
`bootstrap_sample_size`	**Input** - The relative sample size to use for bootstrapping.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`divisor`	**Input** - A value to divide the statistic by.
`id`	The unique id of this protocol.
`observable`	**Input** - The file path to the observable which should be averaged.
`outputs`	A dictionary of the outputs of this property.
`potential_energies`	**Input** - The potential energies which were evaluated at the same configurations and using the same force field parameters as the observable to average.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.
`thermodynamic_state`	**Input** - The state at which the observables were computed.

continues on next page

Table 259 – continued from previous page

time_series_statistics	**Output** - Statistics about the observables from which the average was computed.These include the statistical inefficiency and the index after which the observables have become stationary (i.e.
value	**Output** - The average value of the observable.

observable

Input - The file path to the observable which should be averaged. The default value of this attribute is not set and must be set by the user..

Type *ObservableArray*

divisor

Input - A value to divide the statistic by. This is useful if a statistic (such as enthalpy) needs to be normalised by the number of molecules. The default value of this attribute is `1.0`.

Type typing.Union[int, float, openff.evaluator.utils.units.Quantity]

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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`

Input - The number of bootstrap iterations to perform. The default value of this attribute is `250`.

Type int

`bootstrap_sample_size`

Input - The relative sample size to use for bootstrapping. The default value of this attribute is `1.0`.

Type float

`can_merge`(*other*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*`list of tuple of str, optional`*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*`str`*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

property outputs

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

potential_energies

Input - The potential energies which were evaluated at the same configurations and using the same force field parameters as the observable to average. This is required to compute ensemble averages of the gradients of the observable with respect to force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type *ObservableArray*

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 required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

thermodynamic_state
Input - The state at which the observables were computed. This is required to compute ensemble averages of the gradients of the observable with respect to force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type *ThermodynamicState*

time_series_statistics
Output - Statistics about the observables from which the average was computed.These include the statistical inefficiency and the index after which the observables have become stationary (i.e. equilibrated). The default value of this attribute is not set and must be set by the user..

Type TimeSeriesStatistics

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

value
Output - The average value of the observable. The default value of this attribute is not set and must be set by the user..

Type *Observable*

AverageDielectricConstant

class `openff.evaluator.protocols.analysis.`**AverageDielectricConstant**(*protocol_id*)
Computes the average value of the dielectric constant from a set of dipole moments (M) and volumes (V) sampled over the course of a molecular simulation such that `eps = 1 + (<M^2> - <M>^2) / (3.0 * eps_0 * <V> * kb * T)` [1]_.

References

[1] **A. Glattli, X. Daura and W. F. van Gunsteren. Derivation of an improved simple** point charge model for liquid water: SPC/A and SPC/L. J. Chem. Phys. 116(22): 9811-9828, 2002

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.

continues on next page

Table 260 – continued from previous page

get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
bootstrap_iterations	**Input** - The number of bootstrap iterations to perform.
bootstrap_sample_size	**Input** - The relative sample size to use for bootstrapping.
dependencies	A list of pointers to the protocols which this protocol takes input from.
dipole_moments	**Input** - The dipole moments of each sampled configuration.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
potential_energies	**Input** - The potential energies which were evaluated at the same configurations and using the same force field parameters as the observable to average.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
thermodynamic_state	**Input** - The state at which the observables were computed.
time_series_statistics	**Output** - Statistics about the observables from which the average was computed.These include the statistical inefficiency and the index after which the observables have become stationary (i.e.
value	**Output** - The average value of the observable.
volumes	**Input** - The volume of each sampled configuration.

dipole_moments

Input - The dipole moments of each sampled configuration. The default value of this attribute is not set and must be set by the user..

Type *ObservableArray*

volumes

Input - The volume of each sampled configuration. The default value of this attribute is not set and must be set by the user..

Type *ObservableArray*

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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`

Input - The number of bootstrap iterations to perform. The default value of this attribute is `250`.

Type int

`bootstrap_sample_size`

Input - The relative sample size to use for bootstrapping. The default value of this attribute is `1.0`.

Type float

`can_merge`(*other*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`potential_energies`

Input - The potential energies which were evaluated at the same configurations and using the same force field parameters as the observable to average. This is required to compute ensemble averages of the gradients of the observable with respect to force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type *ObservableArray*

`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 required_inputs`

The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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.

`thermodynamic_state`

Input - The state at which the observables were computed. This is required to compute ensemble averages of the gradients of the observable with respect to force field parameters. The default value of this attribute is not set. This attribute is *optional*.

Type *ThermodynamicState*

`time_series_statistics`

Output - Statistics about the observables from which the average was computed.These include the statistical inefficiency and the index after which the observables have become stationary (i.e. equilibrated). The default value of this attribute is not set and must be set by the user..

Type TimeSeriesStatistics

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

`value`

Output - The average value of the observable. The default value of this attribute is not set and must be set by the user..

Type *Observable*

AverageFreeEnergies

`class openff.evaluator.protocols.analysis.`**`AverageFreeEnergies`**`(`*protocol_id*`)`

A protocol which computes the Boltzmann weighted average (G° = -RT × Log[_{n} exp(-G°_{n})]) of a set of free energies which were measured at the same thermodynamic state. Confidence intervals are computed by bootstrapping with replacement.

`__init__``(`*protocol_id*`)`

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
bootstrap_cycles	**Input** - The number of bootstrap cycles to perform when estimating the uncertainty in the combined free energies.
confidence_intervals	**Output** - The 95% confidence intervals on the average free energy.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
required_inputs	The inputs which must be set on this protocol.
result	**Output** - The sum of the values.
schema	A serializable schema for this object.
thermodynamic_state	**Input** - The thermodynamic state at which the free energies were measured.
values	**Input** - The values to add together.

values: List[*openff.evaluator.utils.observables.Observable*]
Input - The values to add together. The default value of this attribute is not set and must be set by the user..

Type list

thermodynamic_state
Input - The thermodynamic state at which the free energies were measured. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

bootstrap_cycles
Input - The number of bootstrap cycles to perform when estimating the uncertainty in the combined free energies. The default value of this attribute is `2000`.

Type int

result
Output - The sum of the values. The default value of this attribute is not set and must be set by the user..

Type *Observable*

confidence_intervals
Output - The 95% confidence intervals on the average free energy. The default value of this attribute is not set and must be set by the user..

Type Quantity

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

allow_merging
Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

ComputeDipoleMoments

class openff.evaluator.protocols.analysis.**ComputeDipoleMoments**(*protocol_id*)
A protocol which will compute the dipole moment for each configuration in a trajectory and for a given parameterized system.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

continues on next page

Table 264 – 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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`dipole_moments`	**Output** - The computed dipole moments.
`gradient_parameters`	**Input** - An optional list of parameters to differentiate the dipole moments with respect to.
`id`	The unique id of this protocol.
`outputs`	A dictionary of the outputs of this property.
`parameterized_system`	**Input** - The parameterized system which encodes the charge on each atom in the system.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.
`trajectory_path`	**Input** - The file path to the trajectory of configurations.

`parameterized_system`

Input - The parameterized system which encodes the charge on each atom in the system. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

`trajectory_path`

Input - The file path to the trajectory of configurations. The default value of this attribute is not set and must be set by the user..

Type str

`gradient_parameters`

Input - An optional list of parameters to differentiate the dipole moments with respect to.

Type list

`dipole_moments`

Output - The computed dipole moments. The default value of this attribute is not set and must be set by the user..

Type *ObservableArray*

`allow_merging`
Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod `from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod `from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod `get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`

The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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.

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

BaseDecorrelateProtocol

`class` `openff.evaluator.protocols.analysis.`**`BaseDecorrelateProtocol`**(*protocol_id*)

An abstract base class for protocols which will subsample a set of data, yielding only equilibrated, uncorrelated data.

`__init__`(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.
`time_series_statistics`	**Input** - Statistics about the data to decorrelate.

time_series_statistics: Union[openff.evaluator.utils.timeseries.TimeSeriesStatistics, List[openff.evaluator.utils.timeseries.TimeSeriesStatistics]]

Input - Statistics about the data to decorrelate. This should include the statistical inefficiency and the index after which the observables have become stationary (i.e. equilibrated). If a list of such statistics are provided it will be assumed that multiple time series which have been joined together are being decorrelated and hence will each be decorrelated separately. The default value of this attribute is not set and must be set by the user..

Type typing.Union[list, openff.evaluator.utils.timeseries.TimeSeriesStatistics]

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **`schema`** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

DecorrelateTrajectory

class openff.evaluator.protocols.analysis.**DecorrelateTrajectory**(*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*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`input_coordinate_file`	**Input** - The file path to the starting coordinates of a trajectory.
`input_trajectory_path`	**Input** - The file path to the trajectory to subsample.
`output_trajectory_path`	**Output** - The file path to the subsampled trajectory.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.
`time_series_statistics`	**Input** - Statistics about the data to decorrelate.

`input_coordinate_file`

Input - The file path to the starting coordinates of a trajectory. The default value of this attribute is not set and must be set by the user..

Type str

`input_trajectory_path`

Input - The file path to the trajectory to subsample. The default value of this attribute is not set and must be set by the user..

Type str

`output_trajectory_path`
Output - The file path to the subsampled trajectory. The default value of this attribute is not set and must be set by the user..

Type str

`allow_merging`
Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **`other`** (`Protocol`) – The protocol to compare against.

- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`
The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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.

`time_series_statistics: Union[openff.evaluator.utils.timeseries.TimeSeriesStatistics, List[openff.evaluator.utils.timeseries.TimeSeriesStatistics]]`
Input - Statistics about the data to decorrelate. This should include the statistical inefficiency and the index after which the observables have become stationary (i.e. equilibrated). If a list of such statistics are provided it will be assumed that multiple time series which have been joined together are being decorrelated and hence will each be decorrelated separately. The default value of this attribute is not set and must be set by the user..

Type typing.Union[list, openff.evaluator.utils.timeseries.TimeSeriesStatistics]

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

DecorrelateObservables

class `openff.evaluator.protocols.analysis.`**`DecorrelateObservables`**(*protocol_id*)

A protocol which will subsample a trajectory of observables, yielding only uncorrelated entries as determined from a provided statistical inefficiency and equilibration time.

`__init__`(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`input_observables`	**Input** - The observables to decorrelate.
`output_observables`	**Output** - The decorrelated observables.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.
`time_series_statistics`	**Input** - Statistics about the data to decorrelate.

`input_observables`

Input - The observables to decorrelate. The default value of this attribute is not set and must be set by the user..

Type typing.Union[*openff.evaluator.utils.observables.ObservableArray*, *openff.evaluator.utils.observables.ObservableFrame*]

`output_observables`

Output - The decorrelated observables. The default value of this attribute is not set and must be set by the user..

Type typing.Union[*openff.evaluator.utils.observables.ObservableArray*, *openff.evaluator.utils.observables.ObservableFrame*]

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters value (*str*) – The uuid to prepend.

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.

time_series_statistics: Union[openff.evaluator.utils.timeseries.TimeSeriesStatistics, List[openff.evaluator.utils.timeseries.TimeSeriesStatistics]]
Input - Statistics about the data to decorrelate. This should include the statistical inefficiency and the index after which the observables have become stationary (i.e. equilibrated). If a list of such statistics are provided it will be assumed that multiple time series which have been joined together are being decorrelated and hence will each be decorrelated separately. The default value of this attribute is not set and must be set by the user..

Type typing.Union[list, openff.evaluator.utils.timeseries.TimeSeriesStatistics]

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

Coordinate Generation

BuildCoordinatesPackmol	Creates a set of 3D coordinates with a specified composition using the PACKMOL package.
SolvateExistingStructure	Solvates a set of 3D coordinates with a specified solvent using the PACKMOL package.
BuildDockedCoordinates	Creates a set of coordinates for a ligand bound to some receptor.

BuildCoordinatesPackmol

class openff.evaluator.protocols.coordinates.**BuildCoordinatesPackmol**(*protocol_id*)

Creates a set of 3D coordinates with a specified composition using the PACKMOL package.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
assigned_residue_names	**Output** - The residue names which were assigned to each of the components.
box_aspect_ratio	**Input** - The aspect ratio of the simulation box.
coordinate_file_path	**Output** - The file path to the created PDB coordinate file.
count_exact_amount	**Input** - Whether components present in an exact amount (i.e.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
mass_density	**Input** - The target density of the created system.

continues on next page

Table 274 – continued from previous page

max_molecules	**Input** - The maximum number of molecules to be added to the system.
output_number_of_molecules	**Output** - The number of molecules in the created system.
output_substance	**Output** - The substance which was built by packmol.
outputs	A dictionary of the outputs of this property.
required_inputs	The inputs which must be set on this protocol.
retain_packmol_files	**Input** - 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	**Input** - The composition of the system to build.
tolerance	**Input** - The packmol distance tolerance in units compatible with angstroms.
verbose_packmol	**Input** - If True, packmol will print verbose information to the logger The default value of this attribute is `False`.

max_molecules

Input - The maximum number of molecules to be added to the system. The default value of this attribute is `1000`.

Type int

count_exact_amount

Input - Whether components present in an exact amount (i.e. defined with an `ExactAmount`) should be considered when apply the maximum number of molecules constraint. This may be set false, for example, when building a separate solvated protein (n = 1) and solvated protein + ligand complex (n = 2) system but wish for both systems to have the same number of solvent molecules. The default value of this attribute is `True`.

Type bool

mass_density

Input - The target density of the created system. The default value of this attribute is `0.95 g / ml`.

Type Quantity

box_aspect_ratio

Input - The aspect ratio of the simulation box. The default value of this attribute is `[1.0, 1.0, 1.0]`.

Type list

substance

Input - The composition of the system to build. The default value of this attribute is not set and must be set by the user..

Type *Substance*

tolerance

Input - The packmol distance tolerance in units compatible with angstroms. The default value of this attribute is `2.0 Å`.

Type Quantity

verbose_packmol

Input - If True, packmol will print verbose information to the logger The default value of this attribute is `False`.

Type bool

`retain_packmol_files`
Input - If True, packmol will not delete all of the temporary files it creates while building the coordinates. The default value of this attribute is `False`.

Type bool

`output_number_of_molecules`
Output - The number of molecules in the created system. This may be less than maximum requested due to rounding of mole fractions The default value of this attribute is not set and must be set by the user..

Type int

`output_substance`
Output - The substance which was built by packmol. This may differ from the input substance for system containing two or more components due to rounding of mole fractions. The mole fractions provided by this output should always be used when weighting values by a mole fraction. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`assigned_residue_names`
Output - The residue names which were assigned to each of the components. Each key corresponds to a component identifier. The default value of this attribute is not set and must be set by the user..

Type dict

`coordinate_file_path`
Output - The file path to the created PDB coordinate file. The default value of this attribute is not set and must be set by the user..

Type str

`allow_merging`
Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (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

id

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

merge(*other*)

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

Parameters **other** (Protocol) – 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]

property outputs
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

SolvateExistingStructure

class openff.evaluator.protocols.coordinates.**SolvateExistingStructure**(*protocol_id*)

Solvates a set of 3D coordinates with a specified solvent using the PACKMOL package.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
assigned_residue_names	**Output** - The residue names which were assigned to each of the components.
box_aspect_ratio	**Input** - The aspect ratio of the simulation box.
center_solute_in_box	**Input** - If *True*, the solute to solvate will be centered in the simulation box.
coordinate_file_path	**Output** - The file path to the created PDB coordinate file.
count_exact_amount	**Input** - Whether components present in an exact amount (i.e.
dependencies	A list of pointers to the protocols which this protocol takes input from.

continues on next page

Table 276 – continued from previous page

id	The unique id of this protocol.
mass_density	**Input** - The target density of the created system.
max_molecules	**Input** - The maximum number of molecules to be added to the system.
output_number_of_molecules	**Output** - The number of molecules in the created system.
output_substance	**Output** - The substance which was built by packmol.
outputs	A dictionary of the outputs of this property.
required_inputs	The inputs which must be set on this protocol.
retain_packmol_files	**Input** - 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	**Input** - A file path to the solute to solvate.
substance	**Input** - The composition of the system to build.
tolerance	**Input** - The packmol distance tolerance in units compatible with angstroms.
verbose_packmol	**Input** - If True, packmol will print verbose information to the logger The default value of this attribute is `False`.

`solute_coordinate_file`

Input - A file path to the solute to solvate. The default value of this attribute is not set and must be set by the user..

Type str

`center_solute_in_box`

Input - If *True*, the solute to solvate will be centered in the simulation box. The default value of this attribute is `True`.

Type bool

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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

`assigned_residue_names`

Output - The residue names which were assigned to each of the components. Each key corresponds to a component identifier. The default value of this attribute is not set and must be set by the user..

Type dict

`box_aspect_ratio`

Input - The aspect ratio of the simulation box. The default value of this attribute is `[1.0, 1.0, 1.0]`.

Type list

`can_merge`(*other*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- `other` (`Protocol`) – The protocol to compare against.
- `path_replacements` (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

`coordinate_file_path`

Output - The file path to the created PDB coordinate file. The default value of this attribute is not set and must be set by the user..

Type str

`count_exact_amount`

Input - Whether components present in an exact amount (i.e. defined with an `ExactAmount`) should be considered when apply the maximum number of molecules constraint. This may be set false, for example, when building a separate solvated protein (n = 1) and solvated protein + ligand complex (n = 2) system but wish for both systems to have the same number of solvent molecules. The default value of this attribute is `True`.

Type bool

`property dependencies`

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

Type list of ProtocolPath

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **`schema`** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`mass_density`

Input - The target density of the created system. The default value of this attribute is `0.95 g / ml`.

Type Quantity

`max_molecules`

Input - The maximum number of molecules to be added to the system. The default value of this attribute is `1000`.

Type int

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`output_number_of_molecules`

Output - The number of molecules in the created system. This may be less than maximum requested due to rounding of mole fractions The default value of this attribute is not set and must be set by the user..

Type int

`output_substance`

Output - The substance which was built by packmol. This may differ from the input substance for system containing two or more components due to rounding of mole fractions. The mole fractions provided by this output should always be used when weighting values by a mole fraction. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`

The inputs which must be set on this protocol.

Type list of ProtocolPath

`retain_packmol_files`

Input - If True, packmol will not delete all of the temporary files it creates while building the coordinates. The default value of this attribute is `False`.

Type bool

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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`

Input - The composition of the system to build. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`tolerance`

Input - The packmol distance tolerance in units compatible with angstroms. The default value of this attribute is `2.0 Å`.

Type Quantity

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`verbose_packmol`

Input - If True, packmol will print verbose information to the logger The default value of this attribute is `False`.

Type bool

BuildDockedCoordinates

class `openff.evaluator.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*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.

continues on next page

Table 277 – continued from previous page

replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

activate_site_location	**Input** - Defines the method by which the activate site is identified.
allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
docked_complex_coordinate_path	**Output** - The file path to the docked ligand-receptor complex.
docked_ligand_coordinate_path	**Output** - The file path to the coordinates of the ligand in it's docked pose, aligned with the initial *receptor_coordinate_file*.
id	The unique id of this protocol.
ligand_residue_name	**Output** - The residue name assigned to the docked ligand.
ligand_substance	**Input** - A substance containing only the ligand to dock.
number_of_ligand_conformers	**Input** - The number of conformers to try and dock into the receptor structure.
outputs	A dictionary of the outputs of this property.
receptor_coordinate_file	**Input** - The file path to the MOL2 coordinates of the receptor molecule.
receptor_residue_name	**Output** - The residue name assigned to the receptor.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.

`class ActivateSiteLocation`(*value*)

An enum which describes the methods by which a receptors activate site(s) is located.

`ligand_substance`

Input - A substance containing only the ligand to dock. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`number_of_ligand_conformers`

Input - The number of conformers to try and dock into the receptor structure. The default value of this attribute is `100`.

Type int

`receptor_coordinate_file`

Input - The file path to the MOL2 coordinates of the receptor molecule. The default value of this attribute is not set and must be set by the user..

Type str

`activate_site_location`

Input - Defines the method by which the activate site is identified. The default value of this attribute is `ActivateSiteLocation.ReceptorCenterOfMass`.

Type *BuildDockedCoordinates.ActivateSiteLocation*

`docked_ligand_coordinate_path`

Output - The file path to the coordinates of the ligand in it's docked pose, aligned with the initial *receptor_coordinate_file*. The default value of this attribute is not set and must be set by the user..

Type str

`docked_complex_coordinate_path`

Output - The file path to the docked ligand-receptor complex. The default value of this attribute is not set and must be set by the user..

Type str

`ligand_residue_name`

Output - The residue name assigned to the docked ligand. The default value of this attribute is not set and must be set by the user..

Type str

`receptor_residue_name`

Output - The residue name assigned to the receptor. The default value of this attribute is not set and must be set by the user..

Type str

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

Force Field Assignment

BaseBuildSystem	The base class for any protocol whose role is to apply a set of force field parameters to a given system.

continues on next page

Table 279 – continued from previous page

BuildSmirnoffSystem	Parametrise a set of molecules with a given smirnoff force field using the OpenFF toolkit.
BuildLigParGenSystem	Parametrise a set of molecules with the OPLS-AA/M force field.
BuildTLeapSystem	Parametrise a set of molecules with an Amber based force field.

BaseBuildSystem

class openff.evaluator.protocols.forcefield.**BaseBuildSystem**(*protocol_id*)

The base class for any protocol whose role is to apply a set of force field parameters to a given system.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
coordinate_file_path	**Input** - The file path to the PDB coordinate file which defines the topology of 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	**Input** - The file path to the force field parameters to assign to the system.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
parameterized_system	**Output** - The parameterized system object.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
substance	**Input** - The composition of the system.

force_field_path

Input - The file path to the force field parameters to assign to the system. The default value of this attribute is not set and must be set by the user..

Type str

coordinate_file_path

Input - The file path to the PDB coordinate file which defines the topology of the system to which the force field parameters will be assigned. The default value of this attribute is not set and must be set by the user..

Type str

substance

Input - The composition of the system. The default value of this attribute is not set and must be set by the user..

Type *Substance*

parameterized_system

Output - The parameterized system object. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (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

id

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **other** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`

The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

`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.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

BuildSmirnoffSystem

class openff.evaluator.protocols.forcefield.**BuildSmirnoffSystem**(*protocol_id*)

Parametrise a set of molecules with a given smirnoff force field using the OpenFF toolkit.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
coordinate_file_path	**Input** - The file path to the PDB coordinate file which defines the topology of 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	**Input** - The file path to the force field parameters to assign to the system.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
parameterized_system	**Output** - The parameterized system object.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
substance	**Input** - The composition of the system.

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **`other`** (`Protocol`) – The protocol to compare against.
- **`path_replacements`** (*`list of tuple of str, optional`*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

`coordinate_file_path`

Input - The file path to the PDB coordinate file which defines the topology of the system to which the force field parameters will be assigned. The default value of this attribute is not set and must be set by the user..

Type str

`property dependencies`

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

Type list of ProtocolPath

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`force_field_path`

Input - The file path to the force field parameters to assign to the system. The default value of this attribute is not set and must be set by the user..

Type str

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **`schema`** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (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

id

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

merge(*other*)

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

Parameters **other** (Protocol) – 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]

property outputs
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

parameterized_system
Output - The parameterized system object. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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
Input - The composition of the system. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (`type of Attribute, optional`) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

BuildLigParGenSystem

class `openff.evaluator.protocols.forcefield.`**`BuildLigParGenSystem`**(*protocol_id*)
Parametrise a set of molecules with the OPLS-AA/M force field. using a LigParGen server.

Notes

This protocol is currently a work in progress and as such has limited functionality compared to the more established *BuildSmirnoffSystem* protocol.

References

[1] Potential energy functions for atomic-level simulations of water and organic and biomolecular systems. Jorgensen, W. L.; Tirado-Rives, J. Proc. Nat. Acad. Sci. USA 2005, 102, 6665-6670

[2] 1.14*CM1A-LBCC: Localized Bond-Charge Corrected CM1A Charges for Condensed-Phase Simulations. Dodda, L. S.; Vilseck, J. Z.; Tirado-Rives, J.; Jorgensen, W. L. J. Phys. Chem. B, 2017, 121 (15), pp 3864-3870

[3] LigParGen web server: An automatic OPLS-AA parameter generator for organic ligands. Dodda, L. S.;Cabeza de Vaca, I.; Tirado-Rives, J.; Jorgensen, W. L. Nucleic Acids Research, Volume 45, Issue W1, 3 July 2017, Pages W331-W336

`__init__`(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.

continues on next page

Table 284 – 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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
coordinate_file_path	**Input** - The file path to the PDB coordinate file which defines the topology of 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	**Input** - The file path to the force field parameters to assign to the system.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
parameterized_system	**Output** - The parameterized system object.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
substance	**Input** - The composition of the system.
water_model	**Input** - The water model to apply, if any water molecules are present.

`class WaterModel`(*value*)

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 full water model support is introduced.

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

coordinate_file_path
Input - The file path to the PDB coordinate file which defines the topology of the system to which the force field parameters will be assigned. The default value of this attribute is not set and must be set by the user..

Type str

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

Type list of ProtocolPath

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.

- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

force_field_path
Input - The file path to the force field parameters to assign to the system. The default value of this attribute is not set and must be set by the user..

Type str

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`parameterized_system`

Output - The parameterized system object. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`

The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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`

Input - The composition of the system. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **`or`** `AssertionError` –

`water_model`

Input - The water model to apply, if any water molecules are present. The default value of this attribute is `WaterModel.TIP3P`.

Type *TemplateBuildSystem.WaterModel*

BuildTLeapSystem

class `openff.evaluator.protocols.forcefield.`**`BuildTLeapSystem`**(*protocol_id*)

Parametrise a set of molecules with an Amber based force field. using the tleap package.

Notes

- This protocol is currently a work in progress and as such has limited functionality compared to the more established *BuildSmirnoffSystem* protocol.
- This protocol requires the optional *ambertools >=19.0* dependency to be installed.

`__init__`(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
charge_backend	**Input** - The backend framework to use to assign partial charges.
coordinate_file_path	**Input** - The file path to the PDB coordinate file which defines the topology of 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	**Input** - The file path to the force field parameters to assign to the system.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
parameterized_system	**Output** - The parameterized system object.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
substance	**Input** - The composition of the system.
water_model	**Input** - The water model to apply, if any water molecules are present.

`class ChargeBackend`(*value*)
The framework to use to assign partial charges.

`charge_backend`
Input - The backend framework to use to assign partial charges.

Type *BuildTLeapSystem.ChargeBackend*

`class WaterModel`(*value*)
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 full water model support is introduced.

`allow_merging`
Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

coordinate_file_path
Input - The file path to the PDB coordinate file which defines the topology of the system to which the force field parameters will be assigned. The default value of this attribute is not set and must be set by the user..

Type str

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

Type list of ProtocolPath

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

force_field_path
Input - The file path to the force field parameters to assign to the system. The default value of this attribute is not set and must be set by the user..

Type str

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (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

id
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **other** (Protocol) – 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]

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`parameterized_system`
Output - The parameterized system object. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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
Input - The composition of the system. The default value of this attribute is not set and must be set by the user..

Type *Substance*

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

water_model
Input - The water model to apply, if any water molecules are present. The default value of this attribute is `WaterModel.TIP3P`.

Type *TemplateBuildSystem.WaterModel*

Gradients

ZeroGradients	Zeros the gradients of an observable with respect to a specified set of force field parameters.

ZeroGradients

class openff.evaluator.protocols.gradients.**ZeroGradients**(*protocol_id*)
Zeros the gradients of an observable with respect to a specified set of force field parameters.

__init__(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`force_field_path`	**Input** - The path to the force field which contains the parameters to differentiate the observable with respect to.
`gradient_parameters`	**Input** - The parameters to zero the gradient with respect to.
`id`	The unique id of this protocol.
`input_observables`	**Input** - The observable to set the gradients of.
`output_observables`	**Output** - The observable with zeroed gradients.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.

`input_observables`

Input - The observable to set the gradients of. The default value of this attribute is not set and must be set by the user..

Type typing.Union[*openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

force_field_path
: **Input** - The path to the force field which contains the parameters to differentiate the observable with respect to. This is many used to get the correct units for the parameters. The default value of this attribute is not set and must be set by the user..

    Type str

gradient_parameters
: **Input** - The parameters to zero the gradient with respect to.

    Type list

output_observables
: **Output** - The observable with zeroed gradients. The default value of this attribute is not set and must be set by the user..

    Type typing.Union[*openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

allow_merging
: **Input** - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

    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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

Groups

ConditionalGroup	A collection of protocols which are to execute until a given condition is met.

ConditionalGroup

class openff.evaluator.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 ProtocolGroup.

Methods

__init__(protocol_id)	Constructs a new ProtocolGroup.
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)	Add protocols to this group.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another ProtocolGroup with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
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.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
conditions	**Input** - The conditions which must be satisfied beforethe group will cleanly exit.
current_iteration	**Output** - The current number of iterations this group has performed while attempting to satisfy the specified conditions.
dependencies	A list of pointers to the protocols which this protocol takes input from.

continues on next page

Table 293 – continued from previous page

id	The unique id of this protocol.
max_iterations	**Input** - The maximum number of iterations to run for to try and satisfy the groups conditions.
outputs	A dictionary of the outputs of this property.
protocols	A dictionary of the protocols in this groups, where the dictionary key is the protocol id, and the value is the protocol itself.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.

class Condition

Defines a specific condition which must be met of the form *left_hand_value* [TYPE] *right_hand_value*, where *[TYPE]* may be less than or greater than.

class Type(*value*)

The available condition types.

left_hand_value

The left-hand value to compare. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Quantity]

right_hand_value

The right-hand value to compare. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Quantity]

type

The right-hand value to compare. The default value of this attribute is `Type.LessThan`.

Type *ConditionalGroup.Condition.Type*

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`conditions`

Input - The conditions which must be satisfied beforethe group will cleanly exit. The default value of this attribute is `[]`.

Type list

`current_iteration`

Output - The current number of iterations this group has performed while attempting to satisfy the specified conditions. This value starts from one. The default value of this attribute is not set and must be set by the user..

Type int

`max_iterations`

Input - The maximum number of iterations to run for to try and satisfy the groups conditions. The default value of this attribute is `100`.

Type int

`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.

`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** (`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

`add_protocols`(**protocols*)

Add protocols to this group.

Parameters **protocols** (`Protocol`) – The protocols to add.

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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

id
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

property outputs
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

property protocols
A dictionary of the protocols in this groups, where the dictionary key is the protocol id, and the value is the protocol itself.

Notes

This property should *not* be altered. Use *add_protocols* to add new protocols to the group.

Type dict of str and Protocol

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.

property required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

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
WeightByMoleFraction	Multiplies a value by the mole fraction of a component in a *Substance*.
FilterSubstanceByRole	A protocol which takes a substance as input, and returns a substance which only contains components whose role match a given criteria.
DummyProtocol	A protocol whose only purpose is to return an input value as an output value.

AddValues

class openff.evaluator.protocols.miscellaneous.**AddValues**(*protocol_id*)

A protocol to add together a list of values.

Notes

The *values* input must either be a list of openff.evaluator.unit.Quantity, a ProtocolPath to a list of openff.evaluator.unit.Quantity, or a list of ProtocolPath which each point to a openff.evaluator.unit.Quantity.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.

continues on next page

Table 295 – continued from previous page

merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
required_inputs	The inputs which must be set on this protocol.
result	**Output** - The sum of the values.
schema	A serializable schema for this object.
values	**Input** - The values to add together.

values

Input - The values to add together. The default value of this attribute is not set and must be set by the user..

Type list

result

Output - The sum of the values. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (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

id

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

merge(*other*)

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

Parameters **other** (Protocol) – 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]

property outputs

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

SubtractValues

class openff.evaluator.protocols.miscellaneous.**SubtractValues**(*protocol_id*)

A protocol to subtract one value from another such that:

result = value_b - value_a

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
required_inputs	The inputs which must be set on this protocol.

continues on next page

Table 298 – continued from previous page

`result`	**Output** - The results of *value_b* - *value_a*.
`schema`	A serializable schema for this object.
`value_a`	**Input** - *value_a* in the formula *result* = *value_b* - *value_a*.
`value_b`	**Input** - *value_b* in the formula *result* = *value_b* - *value_a*.

value_a

Input - *value_a* in the formula *result* = *value_b* - *value_a*. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

value_b

Input - *value_b* in the formula *result* = *value_b* - *value_a*. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

result

Output - The results of *value_b* - *value_a*. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (`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

id
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

merge(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **other** (`Protocol`) – 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]

property outputs
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

MultiplyValue

class openff.evaluator.protocols.miscellaneous.**MultiplyValue**(*protocol_id*)

A protocol which multiplies a value by a specified scalar

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
multiplier	**Input** - The scalar to multiply by.
outputs	A dictionary of the outputs of this property.
required_inputs	The inputs which must be set on this protocol.
result	**Output** - The result of the multiplication.
schema	A serializable schema for this object.
value	**Input** - The value to multiply.

value

Input - The value to multiply. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

`multiplier`
Input - The scalar to multiply by. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Quantity]

`result`
Output - The result of the multiplication. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

`allow_merging`
Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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** (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

id

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

merge(*other*)

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

Parameters **other** (Protocol) – 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]

property outputs

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs`
The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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.

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

DivideValue

`class` `openff.evaluator.protocols.miscellaneous.`**`DivideValue`**(*protocol_id*)
A protocol which divides a value by a specified scalar

`__init__`(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`divisor`	**Input** - The scalar to divide by.
`id`	The unique id of this protocol.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`result`	**Output** - The result of the division.
`schema`	A serializable schema for this object.
`value`	**Input** - The value to divide.

value

Input - The value to divide. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

divisor

Input - The scalar to divide by. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Quantity]

result

Output - The result of the division. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

WeightByMoleFraction

class openff.evaluator.protocols.miscellaneous.**WeightByMoleFraction**(*protocol_id*)
Multiplies a value by the mole fraction of a component in a *Substance*.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.

continues on next page

Table 303 – continued from previous page

from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
component	**Input** - The component whose mole fraction to weight by.
dependencies	A list of pointers to the protocols which this protocol takes input from.
full_substance	**Input** - The full substance which describes the mole fraction of the component.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
value	**Input** - The value to be weighted.
weighted_value	**Output** - The value weighted by the *component`s mole fraction as determined from the `full_substance*.

value

Input - The value to be weighted. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

component

Input - The component whose mole fraction to weight by. The default value of this attribute is not set and must be set by the user..

Type *Substance*

full_substance

Input - The full substance which describes the mole fraction of the component. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`weighted_value`

Output - The value weighted by the *component`s mole fraction as determined from the `full_substance*. The default value of this attribute is not set and must be set by the user..

Type typing.Union[int, float, openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*]

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.

- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (`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

id

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

merge(*other*)

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

Parameters **other** (`Protocol`) – 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]

property outputs

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

FilterSubstanceByRole

class openff.evaluator.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*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`component_roles`	**Input** - The roles to filter substance components against.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`expected_components`	**Input** - The number of components expected to remain after filtering.
`filtered_substance`	**Output** - The filtered substance.
`id`	The unique id of this protocol.
`input_substance`	**Input** - The substance to filter.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.

`input_substance`

Input - The substance to filter. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`component_roles`

Input - The roles to filter substance components against. The default value of this attribute is not set and must be set by the user..

Type list

`expected_components`

Input - The number of components expected to remain after filtering. An exception is raised if this number is not matched. The default value of this attribute is not set. This attribute is *optional*.

Type int

`filtered_substance`

Output - The filtered substance. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

DummyProtocol

class openff.evaluator.protocols.miscellaneous.**DummyProtocol**(*protocol_id*)
A protocol whose only purpose is to return an input value as an output value.

__init__(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`input_value`	**Input** - A dummy input.
`output_value`	**Output** - A dummy output.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.

`input_value`

Input - A dummy input. The default value of this attribute is not set and must be set by the user..

Type typing.Union[str, int, float, openff.evaluator.utils.units.Quantity, openff.evaluator.utils.units.Measurement, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.forcefield.gradients.ParameterGradientKey*, list, tuple, dict, set, frozenset]

`output_value`

Output - A dummy output. The default value of this attribute is not set and must be set by the user..

Type typing.Union[str, int, float, openff.evaluator.utils.units.Quantity, openff.evaluator.utils.units.Measurement, *openff.evaluator.utils.observables.Observable*, *openff.evaluator.utils.observables.ObservableArray*, *openff.evaluator.forcefield.gradients.ParameterGradient*, *openff.evaluator.forcefield.gradients.ParameterGradientKey*, list, tuple, dict, set, frozenset]

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`

The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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.

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

OpenMM

`OpenMMEnergyMinimisation`	A protocol to minimise the potential energy of a system using OpenMM.
`OpenMMSimulation`	Performs a molecular dynamics simulation in a given ensemble using an OpenMM backend.
`OpenMMEvaluateEnergies`	Re-evaluates the energy of a series of configurations for a given set of force field parameters using OpenMM.

OpenMMEnergyMinimisation

class openff.evaluator.protocols.openmm.**OpenMMEnergyMinimisation**(*protocol_id*)

A protocol to minimise the potential energy of a system using OpenMM.

__init__(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`enable_pbc`	**Input** - If true, periodic boundary conditions will be enabled.
`id`	The unique id of this protocol.
`input_coordinate_file`	**Input** - The coordinates to minimise.
`max_iterations`	**Input** - The maximum number of iterations to perform.
`output_coordinate_file`	**Output** - The file path to the minimised coordinates.
`outputs`	A dictionary of the outputs of this property.

continues on next page

Table 311 – continued from previous page

parameterized_system	**Input** - The parameterized system object which encodes the systems potential energy function.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
tolerance	**Input** - The energy tolerance to which the system should be minimized.

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.

- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`enable_pbc`
Input - If true, periodic boundary conditions will be enabled. The default value of this attribute is `True`.

Type bool

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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** (`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

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`input_coordinate_file`
Input - The coordinates to minimise. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`max_iterations`
Input - 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. The default value of this attribute is `0`.

Type int

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **other** (`Protocol`) – 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]

`output_coordinate_file`
Output - The file path to the minimised coordinates. The default value of this attribute is not set and must be set by the user..

Type str

property outputs
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

parameterized_system
Input - The parameterized system object which encodes the systems potential energy function. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

tolerance
Input - The energy tolerance to which the system should be minimized. The default value of this attribute is `10.0 kJ / mol`.

Type Quantity

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

OpenMMSimulation

class `openff.evaluator.protocols.openmm.OpenMMSimulation`(*protocol_id*)

Performs a molecular dynamics simulation in a given ensemble using an OpenMM backend.

This protocol employs the Langevin integrator implemented in the `openmmtools` package to propagate the state of the system using the default BAOAB splitting [1]_. Further, simulations which are run in the NPT simulation will have a Monte Carlo barostat (openmm.MonteCarloBarostat) applied every 25 steps (the OpenMM default).

References

[1] Leimkuhler, Ben, and Charles Matthews. "Numerical methods for stochastic molecular dynamics." Molecular Dynamics. Springer, Cham, 2015. 261-328.

`__init__`(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_gpu_platforms	**Input** - If true, the simulation will be performed using a GPU if available, otherwise it will be constrained to only using CPUs.
allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
checkpoint_frequency	**Input** - The frequency (in multiples of *output_frequency*) with which to write to a checkpoint file, e.g.
dependencies	A list of pointers to the protocols which this protocol takes input from.
enable_pbc	**Input** - If true, periodic boundary conditions will be enabled.
ensemble	**Input** - The thermodynamic ensemble to simulate in.
gradient_parameters	**Input** - An optional list of parameters to differentiate the evaluated energies with respect to.
high_precision	**Input** - If true, the simulation will be run using double precision.
id	The unique id of this protocol.
input_coordinate_file	**Input** - The file path to the starting coordinates.
observables	**Output** - The observables collected during the simulation.
output_coordinate_file	**Output** - The file path to the coordinates of the final system configuration.
output_frequency	**Input** - The frequency (in number of steps) with which to write to the output statistics and trajectory files.
outputs	A dictionary of the outputs of this property.
parameterized_system	**Input** - The parameterized system object which encodes the systems potential energy function.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
steps_per_iteration	**Input** - The number of steps to propogate the system by at each iteration.
thermodynamic_state	**Input** - The thermodynamic conditions to simulate under The default value of this attribute is not set and must be set by the user..
thermostat_friction	**Input** - The thermostat friction coefficient.
timestep	**Input** - The timestep to evolve the system by at each step.
total_number_of_iterations	**Input** - The number of times to propogate the system forward by the *steps_per_iteration* number of steps.
trajectory_file_path	**Output** - The file path to the trajectory sampled during the simulation.

allow_gpu_platforms

Input - If true, the simulation will be performed using a GPU if available, otherwise it will be constrained to only using CPUs. The default value of this attribute is `True`.

Type bool

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

`checkpoint_frequency`

Input - The frequency (in multiples of *output_frequency*) with which to write to a checkpoint file, e.g. if *output_frequency=100* and *checkpoint_frequency==2*, a checkpoint file would be saved every 200 steps.

When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `10`. This attribute is *optional*.

Type int

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

Type list of ProtocolPath

`enable_pbc`
Input - If true, periodic boundary conditions will be enabled. The default value of this attribute is `True`.

Type bool

`ensemble`
Input - The thermodynamic ensemble to simulate in. The default value of this attribute is `Ensemble.NPT`.

Type Ensemble

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **`schema`** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`gradient_parameters`
Input - An optional list of parameters to differentiate the evaluated energies with respect to.

Type list

`high_precision`
Input - If true, the simulation will be run using double precision. The default value of this attribute is `False`.

Type bool

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`input_coordinate_file`
Input - The file path to the starting coordinates. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

`observables`
Output - The observables collected during the simulation. The default value of this attribute is not set and must be set by the user..

Type *ObservableFrame*

`output_coordinate_file`
Output - The file path to the coordinates of the final system configuration. The default value of this attribute is not set and must be set by the user..

Type str

`output_frequency`
Input - The frequency (in number of steps) with which to write to the output statistics and trajectory files. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `3000`.

Type int

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`parameterized_system`
Input - The parameterized system object which encodes the systems potential energy function. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`
The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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`
Input - The number of steps to propogate the system by at each iteration. The total number of steps performed by this protocol will be *total_number_of_iterations * steps_per_iteration*. The default value of this attribute is `1000000`.

Type int

`thermodynamic_state`
Input - The thermodynamic conditions to simulate under The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

`thermostat_friction`
Input - The thermostat friction coefficient. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `1.0 / ps`.

Type Quantity

`timestep`
Input - The timestep to evolve the system by at each step. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `2.0 fs`.

Type Quantity

`total_number_of_iterations`
Input - The number of times to propogate the system forward by the *steps_per_iteration* number of steps. The total number of steps performed by this protocol will be *total_number_of_iterations * steps_per_iteration*. The default value of this attribute is `1`.

Type int

`trajectory_file_path`
Output - The file path to the trajectory sampled during the simulation. The default value of this attribute is not set and must be set by the user..

Type str

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

OpenMMEvaluateEnergies

class `openff.evaluator.protocols.openmm.`**`OpenMMEvaluateEnergies`**(*protocol_id*)

Re-evaluates the energy of a series of configurations for a given set of force field parameters using OpenMM.

`__init__`(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`enable_pbc`	**Input** - If true, periodic boundary conditions will be enabled.
`gradient_parameters`	**Input** - An optional list of parameters to differentiate the evaluated energies with respect to.
`id`	The unique id of this protocol.
`output_observables`	**Output** - An observable array which stores the reduced potentials potential energies evaluated at the specified state and using the specified system object for each configuration in the trajectory.

continues on next page

Table 315 – continued from previous page

outputs	A dictionary of the outputs of this property.
parameterized_system	**Input** - The parameterized system object which encodes the systems potential energy function.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
thermodynamic_state	**Input** - The state to calculate the reduced potentials at.
trajectory_file_path	**Input** - The path to the trajectory file which contains the configurations to calculate the energies of.

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

enable_pbc
Input - If true, periodic boundary conditions will be enabled. The default value of this attribute is `True`.

Type bool

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (`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

`gradient_parameters`
Input - An optional list of parameters to differentiate the evaluated energies with respect to.

Type list

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **other** (`Protocol`) – 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]

`output_observables`
Output - An observable array which stores the reduced potentials potential energies evaluated at the specified state and using the specified system object for each configuration in the trajectory. The default value of this attribute is not set and must be set by the user..

Type *ObservableFrame*

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

parameterized_system
: **Input** - The parameterized system object which encodes the systems potential energy function. The default value of this attribute is not set and must be set by the user..

  Type ParameterizedSystem

classmethod parse_json(*string_contents*)
: Parses a typed json string into the corresponding class structure.

  Parameters **string_contents** (*str or bytes*) – The typed json string.

  Returns The parsed class.

  Return type Any

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 required_inputs
: The inputs which must be set on this protocol.

  Type list of ProtocolPath

property schema
: A serializable schema for this object.

  Type *ProtocolSchema*

set_uuid(*value*)
: Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

  Parameters **value** (*str*) – The uuid to prepend.

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.

thermodynamic_state
: **Input** - The state to calculate the reduced potentials at. The default value of this attribute is not set and must be set by the user..

  Type *ThermodynamicState*

trajectory_file_path
: **Input** - The path to the trajectory file which contains the configurations to calculate the energies of. The default value of this attribute is not set and must be set by the user..

Type str

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

Paprika

PreparePullCoordinates	A protocol which will align a host-guest complex to the z-axis and position the guest molecule at a specified point along the pull axis.
PrepareReleaseCoordinates	A protocol which will extract the host molecule from a file containing both the host and guest molecules and produce a coordinate file containing only the host which has been correctly aligned to the z-axis.
AddDummyAtoms	A protocol which will add the reference 'dummy' atoms to a parameterised system.

PreparePullCoordinates

class openff.evaluator.protocols.paprika.coordinates.**PreparePullCoordinates**(*protocol_id*)

A protocol which will align a host-guest complex to the z-axis and position the guest molecule at a specified point along the pull axis.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.

continues on next page

Table 317 – continued from previous page

replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
complex_file_path	**Input** - The path to the file which the coordinates of the guest moleculebound to the host molecule.
dependencies	A list of pointers to the protocols which this protocol takes input from.
guest_orientation_mask	**Input** - The string mask which describes which guest atoms will be restrained relative to the dummy atoms to keep the molecule aligned to the z-axis.
id	The unique id of this protocol.
n_pull_windows	**Input** - The total number of the pull windows in the calculation.
output_coordinate_path	**Output** - The file path to the system which has been correctly aligned to the z-axis.
outputs	A dictionary of the outputs of this property.
pull_distance	**Input** - The total distance that the guest will be pulled along the z-axis during the pull phase.
pull_window_index	**Input** - The index of the pull window to generate co-ordinates for.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
substance	**Input** - The substance which defines the host, guest and solvent.

guest_orientation_mask

Input - The string mask which describes which guest atoms will be restrained relative to the dummy atoms to keep the molecule aligned to the z-axis. This should be of the form 'X Y' where X Y are ParmEd selectors for the first and second guest atoms. The default value of this attribute is not set and must be set by the user..

Type str

pull_distance

Input - The total distance that the guest will be pulled along the z-axis during the pull phase. The default value of this attribute is not set and must be set by the user..

Type Quantity

pull_window_index

Input - The index of the pull window to generate coordinates for. The default value of this attribute is not set and must be set by the user..

Type int

n_pull_windows

Input - The total number of the pull windows in the calculation. The default value of this attribute is not set and must be set by the user..

Type int

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

`complex_file_path`

Input - The path to the file which the coordinates of the guest moleculebound to the host molecule. The default value of this attribute is not set and must be set by the user..

Type str

`property dependencies`

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

Type list of ProtocolPath

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **`schema`** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

`output_coordinate_path`
Output - The file path to the system which has been correctly aligned to the z-axis. The default value of this attribute is not set and must be set by the user..

Type str

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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
Input - The substance which defines the host, guest and solvent. The default value of this attribute is not set and must be set by the user..

Type *Substance*

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

PrepareReleaseCoordinates

class `openff.evaluator.protocols.paprika.coordinates.`**`PrepareReleaseCoordinates`**(*protocol_id*)

A protocol which will extract the host molecule from a file containing both the host and guest molecules and produce a coordinate file containing only the host which has been correctly aligned to the z-axis.

`__init__`(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`complex_file_path`	**Input** - The path to the file which the coordinates of the guest moleculebound to the host molecule.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`output_coordinate_path`	**Output** - The file path to the system which has been correctly aligned to the z-axis.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.

continues on next page

Table 320 – continued from previous page

substance	**Input** - The substance which defines the host, guest and solvent.

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

`complex_file_path`
: **Input** - The path to the file which the coordinates of the guest moleculebound to the host molecule. The default value of this attribute is not set and must be set by the user..

  Type str

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

  Type list of ProtocolPath

`execute`(*directory=''*, *available_resources=None*)
: Execute the protocol.

  Parameters

  - `directory` (*str*) – The directory to store output data in.
  - `available_resources` (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
: Create this object from a JSON file.

  Parameters `file_path` (*str*) – The path to load the JSON from.

  Returns The parsed class.

  Return type cls

`classmethod from_schema`(*schema*)
: Initializes a protocol from it's schema definition.

  Parameters `schema` (`ProtocolSchema`) – The schema to initialize the protocol using.

  Returns The initialized protocol.

  Return type cls

`classmethod get_attributes`(*attribute_type=None*)
: Returns all attributes of a specific *attribute_type*.

  Parameters `attribute_type` (*type of Attribute, optional*) – The type of attribute to search for.

  Returns The names of the attributes of the specified type.

  Return type list of str

`get_class_attribute`(*reference_path*)
: Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

  Returns The class attribute.

  Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`output_coordinate_path`

Output - The file path to the system which has been correctly aligned to the z-axis. The default value of this attribute is not set and must be set by the user..

Type str

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`

The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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`

Input - The substance which defines the host, guest and solvent. The default value of this attribute is not set and must be set by the user..

Type *Substance*

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises **`ValueError` or `AssertionError`** –

AddDummyAtoms

class openff.evaluator.protocols.paprika.coordinates.**AddDummyAtoms**(*protocol_id*)

A protocol which will add the reference 'dummy' atoms to a parameterised system. This protocol assumes the host / complex has already been correctly aligned to the z-axis and has been placed at the origin.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
input_coordinate_path	**Input** - The file path to the coordinates which the dummy atoms should be added to.
input_system	**Input** - The parameterized system which the dummy atoms should be added to.
offset	**Input** - The distance to offset the dummy atoms from the origin (0, 0, 0) backwards along the z-axis.

continues on next page

Table 322 – continued from previous page

output_coordinate_path	**Output** - The file path to the coordinates which include the added dummy atoms.
output_system	**Output** - The parameterized system which include the added dummy atoms.
outputs	A dictionary of the outputs of this property.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
substance	**Input** - The substance which defines the host, guest and solvent.

substance

Input - The substance which defines the host, guest and solvent. The default value of this attribute is not set and must be set by the user..

Type *Substance*

offset

Input - The distance to offset the dummy atoms from the origin (0, 0, 0) backwards along the z-axis. The default value of this attribute is not set and must be set by the user..

Type Quantity

input_coordinate_path

Input - The file path to the coordinates which the dummy atoms should be added to. The default value of this attribute is not set and must be set by the user..

Type str

input_system

Input - The parameterized system which the dummy atoms should be added to. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

output_coordinate_path

Output - The file path to the coordinates which include the added dummy atoms. The default value of this attribute is not set and must be set by the user..

Type str

output_system

Output - The parameterized system which include the added dummy atoms. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (`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

id
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

merge(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **other** (Protocol) – 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]

property outputs
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

GenerateAttachRestraints	Generates the restraint values to apply during the 'attach' phase from a set of restraint schema definitions and makes them easily accessible for the protocols which will apply them to the parameterized system.
GeneratePullRestraints	Generates the restraint values to apply during the 'pull' phase from a set of restraint schema definitions and makes them easily accessible for the protocols which will apply them to the parameterized system.
GenerateReleaseRestraints	Generates the restraint values to apply during the 'release' phase from a set of restraint schema definitions and makes them easily accessible for the protocols which will apply them to the parameterized system.
ApplyRestraints	A protocol which will apply the restraints defined in a restraints JSON file to a specified system.

GenerateAttachRestraints

class openff.evaluator.protocols.paprika.restraints.**GenerateAttachRestraints**(*protocol_id*)

Generates the restraint values to apply during the 'attach' phase from a set of restraint schema definitions and makes them easily accessible for the protocols which will apply them to the parameterized system.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.

continues on next page

Table 324 – 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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`attach_lambdas`	**Input** - The values of lambda to use for the attach phase.
`complex_coordinate_path`	**Input** - The file path to a coordinate file which contains the solvatedhost-guest complex and has the anchor dummy atoms added.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`restraint_schemas`	**Input** - The full set of restraint schemas.
`restraints_path`	**Output** - The file path to the *paprika* generated restraints JSON file.
`schema`	A serializable schema for this object.

complex_coordinate_path

Input - The file path to a coordinate file which contains the solvatedhost-guest complex and has the anchor dummy atoms added. The default value of this attribute is not set and must be set by the user..

Type str

attach_lambdas

Input - The values of lambda to use for the attach phase. These muststart from 0.0 and increase monotonically to and include 1.0. The default value of this attribute is not set and must be set by the user..

Type list

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (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

id

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str`, `optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str` or `bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`
The inputs which must be set on this protocol.

Type list of ProtocolPath

`restraint_schemas`
Input - The full set of restraint schemas. The default value of this attribute is not set and must be set by the user..

Type dict

`restraints_path`
Output - The file path to the *paprika* generated restraints JSON file. The default value of this attribute is not set and must be set by the user..

Type str

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (`str`) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

GeneratePullRestraints

class openff.evaluator.protocols.paprika.restraints.**GeneratePullRestraints**(*protocol_id*)
Generates the restraint values to apply during the 'pull' phase from a set of restraint schema definitions and makes them easily accessible for the protocols which will apply them to the parameterized system.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.

continues on next page

Table 326 – continued from previous page

`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`attach_lambdas`	**Input** - The values of lambda to use for the attach phase.
`complex_coordinate_path`	**Input** - The file path to a coordinate file which contains the solvatedhost-guest complex and has the anchor dummy atoms added.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`n_pull_windows`	**Input** - The number of lambda to use for the pull phase.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`restraint_schemas`	**Input** - The full set of restraint schemas.
`restraints_path`	**Output** - The file path to the *paprika* generated restraints JSON file.
`schema`	A serializable schema for this object.

n_pull_windows

Input - The number of lambda to use for the pull phase. The default value of this attribute is not set and must be set by the user..

Type int

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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

`attach_lambdas`

Input - The values of lambda to use for the attach phase. These muststart from 0.0 and increase monotonically to and include 1.0. The default value of this attribute is not set and must be set by the user..

Type list

`can_merge`(*other*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

`complex_coordinate_path`

Input - The file path to a coordinate file which contains the solvatedhost-guest complex and has the anchor dummy atoms added. The default value of this attribute is not set and must be set by the user..

Type str

`property dependencies`

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

Type list of ProtocolPath

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **`schema`** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

restraint_schemas

Input - The full set of restraint schemas. The default value of this attribute is not set and must be set by the user..

Type dict

restraints_path

Output - The file path to the *paprika* generated restraints JSON file. The default value of this attribute is not set and must be set by the user..

Type str

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

GenerateReleaseRestraints

class openff.evaluator.protocols.paprika.restraints.**GenerateReleaseRestraints**(*protocol_id*)

Generates the restraint values to apply during the 'release' phase from a set of restraint schema definitions and makes them easily accessible for the protocols which will apply them to the parameterized system.

__init__(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`host_coordinate_path`	**Input** - The file path to a coordinate file which contains the solvatedhost molecule and has the anchor dummy atoms added.
`id`	The unique id of this protocol.
`outputs`	A dictionary of the outputs of this property.
`release_lambdas`	**Input** - The values of lambda to use for the release phase.
`required_inputs`	The inputs which must be set on this protocol.

continues on next page

Table 329 – continued from previous page

restraint_schemas	**Input** - The full set of restraint schemas.
restraints_path	**Output** - The file path to the *paprika* generated restraints JSON file.
schema	A serializable schema for this object.

host_coordinate_path

Input - The file path to a coordinate file which contains the solvatedhost molecule and has the anchor dummy atoms added. The default value of this attribute is not set and must be set by the user..

Type str

release_lambdas

Input - The values of lambda to use for the release phase. These muststart from 1.0 and decrease monotonically to and include 0.0. The default value of this attribute is not set and must be set by the user..

Type list

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`

The inputs which must be set on this protocol.

Type list of ProtocolPath

`restraint_schemas`

Input - The full set of restraint schemas. The default value of this attribute is not set and must be set by the user..

Type dict

`restraints_path`

Output - The file path to the *paprika* generated restraints JSON file. The default value of this attribute is not set and must be set by the user..

Type str

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

`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.

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

ApplyRestraints

class openff.evaluator.protocols.paprika.restraints.**ApplyRestraints**(*protocol_id*)

A protocol which will apply the restraints defined in a restraints JSON file to a specified system.

__init__(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`load_restraints`(file_path)	Loads a set of *paprika* restraint objects from a JSON file.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`input_system`	**Input** - The parameterized system which the restraints should be added to.
`output_system`	**Output** - The parameterized system which now includes the added restraints.
`outputs`	A dictionary of the outputs of this property.
`phase`	**Input** - The APR phase to take the restraints from.

continues on next page

Table 331 – continued from previous page

`required_inputs`	The inputs which must be set on this protocol.
`restraints_path`	**Input** - The file path to the JSON file which contains the restraint definitions.
`schema`	A serializable schema for this object.
`window_index`	**Input** - The index of the window to take the restraints from.

`restraints_path`

Input - The file path to the JSON file which contains the restraint definitions. This will usually have been generated by a *GenerateXXXRestraints* protocol. The default value of this attribute is not set and must be set by the user..

Type str

`phase`

Input - The APR phase to take the restraints from. The default value of this attribute is not set and must be set by the user..

Type str

`window_index`

Input - The index of the window to take the restraints from. The default value of this attribute is not set and must be set by the user..

Type int

`input_system`

Input - The parameterized system which the restraints should be added to. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

`output_system`

Output - The parameterized system which now includes the added restraints. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

`classmethod load_restraints`(*file_path: str*)

Loads a set of *paprika* restraint objects from a JSON file.

Parameters **`file_path`** – The path to the JSON serialized restraints.

Returns

Return type The loaded *paprika* restraint objects.

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema(schema)`

Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes(attribute_type=None)`

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (`type of Attribute, optional`) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute(reference_path)`

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json(file_path=None, format=False)`

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

merge(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **other** (Protocol) – 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]

property outputs
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

AnalyzeAPRPhase	A protocol which will analyze the outputs of the attach, pull or release phases of an APR calculation and return the change in free energy for that phase of the calculation.
ComputeSymmetryCorrection	Computes the symmetry correction for an APR calculation which involves a guest with symmetry.
ComputeReferenceWork	Computes the reference state work.

AnalyzeAPRPhase

class openff.evaluator.protocols.paprika.analysis.**AnalyzeAPRPhase**(*protocol_id*)

A protocol which will analyze the outputs of the attach, pull or release phases of an APR calculation and return the change in free energy for that phase of the calculation.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol

continues on next page

Table 333 – continued from previous page

set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
phase	**Input** - The phase of the calculation being analyzed.
required_inputs	The inputs which must be set on this protocol.
restraints_path	**Input** - The file path to the JSON file which contains the restraint definitions.
result	**Output** - The analysed free energy.
schema	A serializable schema for this object.
topology_path	**Input** - The file path to a coordinate file which contains topological information about the system.
trajectory_paths	**Input** - A list of paths to the trajectories (in the correct order) generated during the phase being analyzed.

topology_path

Input - The file path to a coordinate file which contains topological information about the system. The default value of this attribute is not set and must be set by the user..

Type str

trajectory_paths

Input - A list of paths to the trajectories (in the correct order) generated during the phase being analyzed. The default value of this attribute is not set and must be set by the user..

Type list

phase

Input - The phase of the calculation being analyzed. The default value of this attribute is not set and must be set by the user..

Type str

restraints_path

Input - The file path to the JSON file which contains the restraint definitions. This will usually have been generated by a *GenerateXXXRestraints* protocol. The default value of this attribute is not set and must be set by the user..

Type str

result

Output - The analysed free energy. The default value of this attribute is not set and must be set by the user..

Type *Observable*

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **`schema`** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters value (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

ComputeSymmetryCorrection

class openff.evaluator.protocols.paprika.analysis.**ComputeSymmetryCorrection**(*protocol_id*)
Computes the symmetry correction for an APR calculation which involves a guest with symmetry.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.

continues on next page

Table 335 – continued from previous page

from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
n_microstates	**Input** - The number of symmetry microstates of the guest molecule.
outputs	A dictionary of the outputs of this property.
required_inputs	The inputs which must be set on this protocol.
result	**Output** - The symmetry correction.
schema	A serializable schema for this object.
thermodynamic_state	**Input** - The thermodynamic state that the calculation was performed at.

`n_microstates`

Input - The number of symmetry microstates of the guest molecule. The default value of this attribute is not set and must be set by the user..

Type int

`thermodynamic_state`

Input - The thermodynamic state that the calculation was performed at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

`result`

Output - The symmetry correction. The default value of this attribute is not set and must be set by the user..

Type *Observable*

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **`directory`** (`str`) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (`str`) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **`schema`** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`
The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

ComputeReferenceWork

`class openff.evaluator.protocols.paprika.analysis.ComputeReferenceWork(protocol_id)`
Computes the reference state work.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.

continues on next page

Table 337 – continued from previous page

`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`restraints_path`	**Input** - The file path to the JSON file which contains the restraint definitions.
`result`	**Output** - The reference state work.
`schema`	A serializable schema for this object.
`thermodynamic_state`	**Input** - The thermodynamic state that the calculation was performed at.

`thermodynamic_state`

Input - The thermodynamic state that the calculation was performed at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

`restraints_path`

Input - The file path to the JSON file which contains the restraint definitions. This will usually have been generated by a *GenerateXXXRestraints* protocol. The default value of this attribute is not set and must be set by the user..

Type str

`result`

Output - The reference state work. The default value of this attribute is not set and must be set by the user..

Type *Observable*

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`
The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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.

`validate`(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

Reweighting

`ConcatenateTrajectories`	A protocol which concatenates multiple trajectories into a single one.
`ConcatenateObservables`	A protocol which concatenates multiple `ObservableFrame` objects into a single `ObservableFrame` object.
`BaseEvaluateEnergies`	A base class for protocols which will re-evaluate the energy of a series of configurations for a given set of force field parameters.
`BaseMBARProtocol`	Re-weights a set of observables using MBAR to calculate the average value of the observables at a different state than they were originally measured.
`ReweightObservable`	Reweight an array of observables to a new state using MBAR.
`ReweightDielectricConstant`	Computes the avergage value of the dielectric constant be re-weighting a set a set of dipole moments and volumes using MBAR.

ConcatenateTrajectories

class openff.evaluator.protocols.reweighting.**ConcatenateTrajectories**(*protocol_id*)

A protocol which concatenates multiple trajectories into a single one.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other 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	**Input** - A list of paths to the starting PDB coordinates for each of the trajectories.
input_trajectory_paths	**Input** - A list of paths to the trajectories to concatenate.
output_coordinate_path	**Output** - The path the PDB coordinate file which contains the topology of the concatenated trajectory.
output_trajectory_path	**Output** - The path to the concatenated trajectory.
outputs	A dictionary of the outputs of this property.

continues on next page

Table 341 – continued from previous page

required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.

input_coordinate_paths

Input - A list of paths to the starting PDB coordinates for each of the trajectories. The default value of this attribute is not set and must be set by the user..

Type list

input_trajectory_paths

Input - A list of paths to the trajectories to concatenate. The default value of this attribute is not set and must be set by the user..

Type list

output_coordinate_path

Output - The path the PDB coordinate file which contains the topology of the concatenated trajectory. The default value of this attribute is not set and must be set by the user..

Type str

output_trajectory_path

Output - The path to the concatenated trajectory. The default value of this attribute is not set and must be set by the user..

Type str

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises **ValueError or AssertionError** –

ConcatenateObservables

class openff.evaluator.protocols.reweighting.**ConcatenateObservables**(*protocol_id*)

A protocol which concatenates multiple `ObservableFrame` objects into a single `ObservableFrame` object.

__init__(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`input_observables`	**Input** - A list of observable arrays to concatenate.
`output_observables`	**Output** - The concatenated observable array.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.

input_observables

Input - A list of observable arrays to concatenate. The default value of this attribute is not set and must be set by the user..

Type list

`output_observables`

Output - The concatenated observable array. The default value of this attribute is not set and must be set by the user..

Type typing.Union[*openff.evaluator.utils.observables.ObservableArray*, *openff.evaluator.utils.observables.ObservableFrame*]

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **`other`** (`Protocol`) – The protocol to compare against.

- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs`

The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*`str`*) – The uuid to prepend.

`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.

`validate`(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

BaseEvaluateEnergies

`class` `openff.evaluator.protocols.reweighting.`**`BaseEvaluateEnergies`**(*protocol_id*)

A base class for protocols which will re-evaluate the energy of a series of configurations for a given set of force field parameters.

`__init__`(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`enable_pbc`	**Input** - If true, periodic boundary conditions will be enabled.
`gradient_parameters`	**Input** - An optional list of parameters to differentiate the evaluated energies with respect to.
`id`	The unique id of this protocol.
`output_observables`	**Output** - An observable array which stores the reduced potentials potential energies evaluated at the specified state and using the specified system object for each configuration in the trajectory.
`outputs`	A dictionary of the outputs of this property.
`parameterized_system`	**Input** - The parameterized system object which encodes the systems potential energy function.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.
`thermodynamic_state`	**Input** - The state to calculate the reduced potentials at.

continues on next page

Table 345 – continued from previous page

trajectory_file_path	**Input** - The path to the trajectory file which contains the configurations to calculate the energies of.

thermodynamic_state

Input - The state to calculate the reduced potentials at. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

parameterized_system

Input - The parameterized system object which encodes the systems potential energy function. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

enable_pbc

Input - If true, periodic boundary conditions will be enabled. The default value of this attribute is `True`.

Type bool

trajectory_file_path

Input - The path to the trajectory file which contains the configurations to calculate the energies of. The default value of this attribute is not set and must be set by the user..

Type str

gradient_parameters

Input - An optional list of parameters to differentiate the evaluated energies with respect to.

Type list

output_observables

Output - An observable array which stores the reduced potentials potential energies evaluated at the specified state and using the specified system object for each configuration in the trajectory. The default value of this attribute is not set and must be set by the user..

Type *ObservableFrame*

allow_merging

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (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

id
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

merge(*other*)

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

Parameters **other** (Protocol) – 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]

property outputs

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

BaseMBARProtocol

class openff.evaluator.protocols.reweighting.**BaseMBARProtocol**(*protocol_id*)

Re-weights 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*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.
set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
bootstrap_iterations	**Input** - The number of bootstrap iterations to perform if bootstraped uncertainties have been requested The default value of this attribute is `250`.
bootstrap_uncertainties	**Input** - If true, bootstrapping will be used to estimated the total uncertainty in the reweighted value.

continues on next page

Table 347 – continued from previous page

dependencies	A list of pointers to the protocols which this protocol takes input from.
effective_samples	**Output** - The number of effective samples which were re-weighted.
frame_counts	**Input** - The number of configurations per reference state.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
reference_reduced_potentials	**Input** - The reduced potentials of each configuration evaluated at each of the reference states.
required_effective_samples	**Input** - The minimum number of effective samples required to be able to reweight the observable.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
target_reduced_potentials	**Input** - The reduced potentials of each configuration evaluated at the target state.
value	**Output** - The re-weighted average value of the observable at the target state.

reference_reduced_potentials: List[*openff.evaluator.utils.observables.ObservableArray*]

Input - The reduced potentials of each configuration evaluated at each of the reference states. The default value of this attribute is not set and must be set by the user..

Type list

target_reduced_potentials

Input - The reduced potentials of each configuration evaluated at the target state. The default value of this attribute is not set and must be set by the user..

Type *ObservableArray*

frame_counts

Input - The number of configurations per reference state. The sum of theseshould equal the length of the `reference_reduced_potentials` and `target_reduced_potentials` input arrays as well any input observable arrays. The default value of this attribute is not set and must be set by the user..

Type list

bootstrap_uncertainties

Input - If true, bootstrapping will be used to estimated the total uncertainty in the reweighted value. The default value of this attribute is `False`.

Type bool

bootstrap_iterations

Input - The number of bootstrap iterations to perform if bootstraped uncertainties have been requested The default value of this attribute is `250`.

Type int

required_effective_samples

Input - The minimum number of effective samples required to be able to reweight the observable. If the effective samples is less than this minimum an exception will be raised. The default value of this attribute is `50`.

Type int

`value`
: **Output** - The re-weighted average value of the observable at the target state. The default value of this attribute is not set and must be set by the user..

  Type *Observable*

`effective_samples`
: **Output** - The number of effective samples which were re-weighted. The default value of this attribute is not set and must be set by the user..

  Type float

`allow_merging`
: **Input** - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

  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*, *path_replacements=None*)
: Determines whether this protocol can be merged with another.

  Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*`list of tuple of str, optional`*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*`str`*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

ReweightObservable

class openff.evaluator.protocols.reweighting.**ReweightObservable**(*protocol_id*)
Reweight an array of observables to a new state using MBAR.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.

continues on next page

Table 348 – continued from previous page

`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`bootstrap_iterations`	**Input** - The number of bootstrap iterations to perform if bootstraped uncertainties have been requested The default value of this attribute is `250`.
`bootstrap_uncertainties`	**Input** - If true, bootstrapping will be used to estimated the total uncertainty in the reweighted value.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`effective_samples`	**Output** - The number of effective samples which were re-weighted.
`frame_counts`	**Input** - The number of configurations per reference state.
`id`	The unique id of this protocol.
`observable`	**Input** - The observables to reweight.
`outputs`	A dictionary of the outputs of this property.
`reference_reduced_potentials`	**Input** - The reduced potentials of each configuration evaluated at each of the reference states.
`required_effective_samples`	**Input** - The minimum number of effective samples required to be able to reweight the observable.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.
`target_reduced_potentials`	**Input** - The reduced potentials of each configuration evaluated at the target state.
`value`	**Output** - The re-weighted average value of the observable at the target state.

observable

Input - The observables to reweight. The array should contain the values of the observable evaluated for of each configuration at the target state. The default value of this attribute is not set and must be set by the user..

Type *ObservableArray*

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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`

Input - The number of bootstrap iterations to perform if bootstraped uncertainties have been requested The default value of this attribute is `250`.

Type int

`bootstrap_uncertainties`

Input - If true, bootstrapping will be used to estimated the total uncertainty in the reweighted value. The default value of this attribute is `False`.

Type bool

`can_merge`(*other*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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_samples`
Output - The number of effective samples which were re-weighted. The default value of this attribute is not set and must be set by the user..

Type float

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`frame_counts`
Input - The number of configurations per reference state. The sum of theseshould equal the length of the `reference_reduced_potentials` and `target_reduced_potentials` input arrays as well any input observable arrays. The default value of this attribute is not set and must be set by the user..

Type list

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`
The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **`other`** (`Protocol`) – 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]

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

reference_reduced_potentials: List[*openff.evaluator.utils.observables.ObservableArray*]

Input - The reduced potentials of each configuration evaluated at each of the reference states. The default value of this attribute is not set and must be set by the user..

Type list

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

Input - The minimum number of effective samples required to be able to reweight the observable. If the effective samples is less than this minimum an exception will be raised. The default value of this attribute is `50`.

Type int

property required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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
Input - The reduced potentials of each configuration evaluated at the target state. The default value of this attribute is not set and must be set by the user..

Type *ObservableArray*

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

value
Output - The re-weighted average value of the observable at the target state. The default value of this attribute is not set and must be set by the user..

Type *Observable*

ReweightDielectricConstant

class openff.evaluator.protocols.reweighting.**ReweightDielectricConstant**(*protocol_id*)
Computes the avergage value of the dielectric constant be re-weighting a set a set of dipole moments and volumes using MBAR.

__init__(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
bootstrap_iterations	**Input** - The number of bootstrap iterations to perform if bootstraped uncertainties have been requested The default value of this attribute is `250`.
bootstrap_uncertainties	**Input** - If true, bootstrapping will be used to estimated the total uncertainty in the reweighted value.
dependencies	A list of pointers to the protocols which this protocol takes input from.
dipole_moments	**Input** - The dipole moments evaluated at reference state's configurationsusing the force field of the target state.
effective_samples	**Output** - The number of effective samples which were re-weighted.
frame_counts	**Input** - The number of configurations per reference state.
id	The unique id of this protocol.
outputs	A dictionary of the outputs of this property.
reference_reduced_potentials	**Input** - The reduced potentials of each configuration evaluated at each of the reference states.
required_effective_samples	**Input** - The minimum number of effective samples required to be able to reweight the observable.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
target_reduced_potentials	**Input** - The reduced potentials of each configuration evaluated at the target state.
thermodynamic_state	**Input** - The thermodynamic state to re-weight to.
value	**Output** - The re-weighted average value of the observable at the target state.
volumes	**Input** - The dipole moments evaluated at reference state's configurationsusing the force field of the target state.

dipole_moments

Input - The dipole moments evaluated at reference state's configurationsusing the force field of the target state. The default value of this attribute is not set and must be set by the user..

Type typing.Union[*openff.evaluator.utils.observables.ObservableArray*, list]

volumes

Input - The dipole moments evaluated at reference state's configurationsusing the force field of the target state. The default value of this attribute is not set and must be set by the user..

Type typing.Union[*openff.evaluator.utils.observables.ObservableArray*, list]

thermodynamic_state

Input - The thermodynamic state to re-weight to. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

bootstrap_uncertainties

Input - If true, bootstrapping will be used to estimated the total uncertainty in the reweighted value. The default value of this attribute is `False`.

Type bool

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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`

Input - The number of bootstrap iterations to perform if bootstraped uncertainties have been requested The default value of this attribute is `250`.

Type int

`can_merge`(*other*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **`other`** (`Protocol`) – The protocol to compare against.

- **`path_replacements`** (*`list of tuple of str, optional`*) – Replacements to make in any value reference protocol paths before comparing for equality.

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_samples`
Output - The number of effective samples which were re-weighted. The default value of this attribute is not set and must be set by the user..

Type float

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`frame_counts`
Input - The number of configurations per reference state. The sum of theseshould equal the length of the `reference_reduced_potentials` and `target_reduced_potentials` input arrays as well any input observable arrays. The default value of this attribute is not set and must be set by the user..

Type list

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **`file_path`** (*`str`*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)
Initializes a protocol from it's schema definition.

Parameters **`schema`** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **`attribute_type`** (*`type of Attribute, optional`*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`reference_reduced_potentials: List[openff.evaluator.utils.observables.ObservableArray]`
Input - The reduced potentials of each configuration evaluated at each of the reference states. The default value of this attribute is not set and must be set by the user..

Type list

`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`
Input - The minimum number of effective samples required to be able to reweight the observable. If the effective samples is less than this minimum an exception will be raised. The default value of this attribute is `50`.

Type int

`property required_inputs`
The inputs which must be set on this protocol.

Type list of ProtocolPath

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **`value`** (*str*) – The uuid to prepend.

`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`
Input - The reduced potentials of each configuration evaluated at the target state. The default value of this attribute is not set and must be set by the user..

Type *ObservableArray*

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

value
Output - The re-weighted average value of the observable at the target state. The default value of this attribute is not set and must be set by the user..

Type *Observable*

Simulation

BaseEnergyMinimisation	A base class for protocols which will minimise the potential energy of a given system.
BaseSimulation	A base class for protocols which will perform a molecular simulation in a given ensemble and at a specified state.

BaseEnergyMinimisation

class openff.evaluator.protocols.simulation.**BaseEnergyMinimisation**(*protocol_id*)
A base class for protocols which will minimise the potential energy of a given system.

__init__(*protocol_id*)

Methods

__init__(protocol_id)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.
from_json(file_path)	Create this object from a JSON file.
from_schema(schema)	Initializes a protocol from it's schema definition.
get_attributes([attribute_type])	Returns all attributes of a specific *attribute_type*.
get_class_attribute(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
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.
json([file_path, format])	Creates a JSON representation of this class.
merge(other)	Merges another Protocol with this one.
parse_json(string_contents)	Parses a typed json string into the corresponding class structure.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Prepend a unique identifier to this protocols id.

continues on next page

Table 353 – continued from previous page

set_value(reference_path, value)	Sets the value of one of this protocols inputs.
validate([attribute_type])	Validate the values of the attributes.

Attributes

allow_merging	**Input** - Defines whether this protocols is allowed to merge with other protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
enable_pbc	**Input** - If true, periodic boundary conditions will be enabled.
id	The unique id of this protocol.
input_coordinate_file	**Input** - The coordinates to minimise.
max_iterations	**Input** - The maximum number of iterations to perform.
output_coordinate_file	**Output** - The file path to the minimised coordinates.
outputs	A dictionary of the outputs of this property.
parameterized_system	**Input** - The parameterized system object which encodes the systems potential energy function.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
tolerance	**Input** - The energy tolerance to which the system should be minimized.

`input_coordinate_file`

Input - The coordinates to minimise. The default value of this attribute is not set and must be set by the user..

Type str

`parameterized_system`

Input - The parameterized system object which encodes the systems potential energy function. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

`tolerance`

Input - The energy tolerance to which the system should be minimized. The default value of this attribute is `10.0 kJ / mol`.

Type Quantity

`max_iterations`

Input - 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. The default value of this attribute is `0`.

Type int

`enable_pbc`

Input - If true, periodic boundary conditions will be enabled. The default value of this attribute is `True`.

Type bool

`output_coordinate_file`

Output - The file path to the minimised coordinates. The default value of this attribute is not set and must be set by the user..

Type str

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters value (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters attribute_type (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

BaseSimulation

class openff.evaluator.protocols.simulation.**BaseSimulation**(*protocol_id*)
A base class for protocols which will perform a molecular simulation in a given ensemble and at a specified state.

__init__(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_gpu_platforms`	**Input** - If true, the simulation will be performed using a GPU if available, otherwise it will be constrained to only using CPUs.
`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`checkpoint_frequency`	**Input** - The frequency (in multiples of *output_frequency*) with which to write to a checkpoint file, e.g.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`enable_pbc`	**Input** - If true, periodic boundary conditions will be enabled.
`ensemble`	**Input** - The thermodynamic ensemble to simulate in.
`gradient_parameters`	**Input** - An optional list of parameters to differentiate the evaluated energies with respect to.
`high_precision`	**Input** - If true, the simulation will be run using double precision.
`id`	The unique id of this protocol.
`input_coordinate_file`	**Input** - The file path to the starting coordinates.

continues on next page

Table 356 – continued from previous page

`observables`	**Output** - The observables collected during the simulation.
`output_coordinate_file`	**Output** - The file path to the coordinates of the final system configuration.
`output_frequency`	**Input** - The frequency (in number of steps) with which to write to the output statistics and trajectory files.
`outputs`	A dictionary of the outputs of this property.
`parameterized_system`	**Input** - The parameterized system object which encodes the systems potential energy function.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.
`steps_per_iteration`	**Input** - The number of steps to propogate the system by at each iteration.
`thermodynamic_state`	**Input** - The thermodynamic conditions to simulate under The default value of this attribute is not set and must be set by the user..
`thermostat_friction`	**Input** - The thermostat friction coefficient.
`timestep`	**Input** - The timestep to evolve the system by at each step.
`total_number_of_iterations`	**Input** - The number of times to propogate the system forward by the *steps_per_iteration* number of steps.
`trajectory_file_path`	**Output** - The file path to the trajectory sampled during the simulation.

steps_per_iteration

Input - The number of steps to propogate the system by at each iteration. The total number of steps performed by this protocol will be *total_number_of_iterations * steps_per_iteration*. The default value of this attribute is `1000000`.

Type int

total_number_of_iterations

Input - The number of times to propogate the system forward by the *steps_per_iteration* number of steps. The total number of steps performed by this protocol will be *total_number_of_iterations * steps_per_iteration*. The default value of this attribute is `1`.

Type int

output_frequency

Input - The frequency (in number of steps) with which to write to the output statistics and trajectory files. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `3000`.

Type int

checkpoint_frequency

Input - The frequency (in multiples of *output_frequency*) with which to write to a checkpoint file, e.g. if *output_frequency=100* and *checkpoint_frequency==2*, a checkpoint file would be saved every 200 steps. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `10`. This attribute is *optional*.

Type int

timestep

Input - The timestep to evolve the system by at each step. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `2.0 fs`.

Type Quantity

`thermodynamic_state`
Input - The thermodynamic conditions to simulate under The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

`ensemble`
Input - The thermodynamic ensemble to simulate in. The default value of this attribute is `Ensemble.NPT`.

Type Ensemble

`thermostat_friction`
Input - The thermostat friction coefficient. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `1.0 / ps`.

Type Quantity

`input_coordinate_file`
Input - The file path to the starting coordinates. The default value of this attribute is not set and must be set by the user..

Type str

`parameterized_system`
Input - The parameterized system object which encodes the systems potential energy function. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

`enable_pbc`
Input - If true, periodic boundary conditions will be enabled. The default value of this attribute is `True`.

Type bool

`allow_gpu_platforms`
Input - If true, the simulation will be performed using a GPU if available, otherwise it will be constrained to only using CPUs. The default value of this attribute is `True`.

Type bool

`high_precision`
Input - If true, the simulation will be run using double precision. The default value of this attribute is `False`.

Type bool

`gradient_parameters`
Input - An optional list of parameters to differentiate the evaluated energies with respect to.

Type list

`output_coordinate_file`
Output - The file path to the coordinates of the final system configuration. The default value of this attribute is not set and must be set by the user..

Type str

`trajectory_file_path`
Output - The file path to the trajectory sampled during the simulation. The default value of this attribute is not set and must be set by the user..

Type str

`observables`

Output - The observables collected during the simulation. The default value of this attribute is not set and must be set by the user..

Type *ObservableFrame*

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **`other`** (`Protocol`) – The protocol to compare against.
- **`path_replacements`** (*`list of tuple of str, optional`*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

execute(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str, optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

Storage

UnpackStoredSimulationData	Loads a *StoredSimulationData* object from disk, and makes its attributes easily accessible to other protocols.

UnpackStoredSimulationData

class openff.evaluator.protocols.storage.**UnpackStoredSimulationData**(*protocol_id*)

Loads a *StoredSimulationData* object from disk, and makes its attributes easily accessible to other protocols.

__init__(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`coordinate_file_path`	**Output** - A path to the stored simulation output coordinates.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`force_field_path`	**Output** - A path to the force field parameters used to generate the stored data.
`id`	The unique id of this protocol.
`observables`	**Output** - The stored observables frame.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.
`simulation_data_path`	**Input** - A list / 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`	**Output** - The substance which was stored.
`thermodynamic_state`	**Output** - The thermodynamic state which was stored.

continues on next page

Table 359 – continued from previous page

total_number_of_molecules	**Output** - The total number of molecules in the stored system.
trajectory_file_path	**Output** - A path to the stored simulation trajectory.

simulation_data_path
Input - A list / 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. The default value of this attribute is not set and must be set by the user..

Type typing.Union[list, tuple]

substance
Output - The substance which was stored. The default value of this attribute is not set and must be set by the user..

Type *Substance*

total_number_of_molecules
Output - The total number of molecules in the stored system. The default value of this attribute is not set and must be set by the user..

Type int

thermodynamic_state
Output - The thermodynamic state which was stored. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

observables
Output - The stored observables frame. The default value of this attribute is not set and must be set by the user..

Type *ObservableFrame*

coordinate_file_path
Output - A path to the stored simulation output coordinates. The default value of this attribute is not set and must be set by the user..

Type str

trajectory_file_path
Output - A path to the stored simulation trajectory. The default value of this attribute is not set and must be set by the user..

Type str

force_field_path
Output - A path to the force field parameters used to generate the stored data. The default value of this attribute is not set and must be set by the user..

Type str

allow_merging
Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)
Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)
Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)
Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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`** (`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

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **`file_path`** (*`str`*, *`optional`*) – The (optional) file path to save the JSON file to.
- **`format`** (*`bool`*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **`other`** (`Protocol`) – 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]

`property outputs`

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **`string_contents`** (*`str or bytes`*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

YANK Free Energies

BaseYankProtocol	An abstract base class for protocols which will performs a set of alchemical free energy simulations using the YANK framework.
LigandReceptorYankProtocol	A protocol for performing ligand-receptor alchemical free energy calculations using the YANK framework.
SolvationYankProtocol	A protocol for estimating the change in free energy upon transferring a solute into a solvent (referred to as solvent 1) from a second solvent (referred to as solvent 2) by performing an alchemical free energy calculation using the YANK framework.

BaseYankProtocol

class `openff.evaluator.protocols.yank.`**`BaseYankProtocol`**(*protocol_id*)

An abstract base class for protocols which will performs a set of alchemical free energy simulations using the YANK framework.

`__init__`(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`checkpoint_interval`	**Input** - The number of iterations between saving YANK checkpoint files.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`free_energy_difference`	**Output** - The estimated free energy difference between the two phases ofinterest.
`gradient_parameters`	**Input** - An optional list of parameters to differentiate the estimated free energy with respect to.
`id`	The unique id of this protocol.

continues on next page

Table 362 – continued from previous page

number_of_equilibration_iterations	**Input** - The number of iterations used for equilibration before production run.
number_of_iterations	**Input** - The number of YANK iterations to perform.
outputs	A dictionary of the outputs of this property.
required_inputs	The inputs which must be set on this protocol.
schema	A serializable schema for this object.
setup_only	**Input** - If true, YANK will only create and validate the setup files, but not actually run any simulations.
steps_per_iteration	**Input** - The number of steps per YANK iteration to perform.
thermodynamic_state	**Input** - The state at which to run the calculations.
timestep	**Input** - The length of the timestep to take.
verbose	**Input** - Controls whether or not to run YANK at high verbosity.

thermodynamic_state

Input - The state at which to run the calculations. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

number_of_equilibration_iterations

Input - The number of iterations used for equilibration before production run. Only post-equilibration iterations are written to file. The default value of this attribute is `1`.

Type int

number_of_iterations

Input - The number of YANK iterations to perform. The default value of this attribute is `5000`.

Type int

steps_per_iteration

Input - The number of steps per YANK iteration to perform. The default value of this attribute is `500`.

Type int

checkpoint_interval

Input - The number of iterations between saving YANK checkpoint files. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `1`.

Type int

timestep

Input - The length of the timestep to take. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `2 fs`.

Type Quantity

verbose

Input - Controls whether or not to run YANK at high verbosity. The default value of this attribute is `False`.

Type bool

setup_only

Input - If true, YANK will only create and validate the setup files, but not actually run any simulations. This argument is mainly only to be used for testing purposes. The default value of this attribute is `False`.

Type bool

gradient_parameters
Input - An optional list of parameters to differentiate the estimated free energy with respect to.

Type list

free_energy_difference
Output - The estimated free energy difference between the two phases ofinterest. The default value of this attribute is not set and must be set by the user..

Type *Observable*

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises ValueError **or** AssertionError –

allow_merging
Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

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

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (`ProtocolSchema`) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (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

id

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

merge(*other*)

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

Parameters **other** (Protocol) – 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]

property outputs

A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs

The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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 openff.evaluator.protocols.yank.**LigandReceptorYankProtocol**(*protocol_id*)

A protocol for performing ligand-receptor alchemical free energy calculations using the YANK framework.

__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[, path_replacements])	Determines whether this protocol can be merged with another.
execute([directory, available_resources])	Execute the protocol.

continues on next page

Table 363 – continued from previous page

`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`apply_restraints`	**Input** - Determines whether the ligand should be explicitly restrained to the receptor in order to stop the ligand from temporarily unbinding.
`checkpoint_interval`	**Input** - The number of iterations between saving YANK checkpoint files.
`complex_electrostatic_lambdas`	**Input** - The list of electrostatic alchemical states that YANK should sample at when calculating the free energy of the ligand in complex with the receptor.
`complex_steric_lambdas`	**Input** - The list of steric alchemical states that YANK should sample at when calculating the free energy of the ligand in complex with the receptor.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`force_field_path`	**Input** - The path to the force field which defines the charge method to use for the calculation.
`free_energy_difference`	**Output** - The estimated free energy difference between the two phases ofinterest.
`gradient_parameters`	**Input** - An optional list of parameters to differentiate the estimated free energy with respect to.
`id`	The unique id of this protocol.
`ligand_electrostatic_lambdas`	**Input** - The list of electrostatic alchemical states that YANK should sample at when calculating the free energy of the solvated ligand.
`ligand_residue_name`	**Input** - The residue name of the ligand.

continues on next page

Table 364 – continued from previous page

ligand_steric_lambdas	**Input** - The list of steric alchemical states that YANK should sample at when calculating the free energy of the solvated ligand.
number_of_equilibration_iterations	**Input** - The number of iterations used for equilibration before production run.
number_of_iterations	**Input** - The number of YANK iterations to perform.
outputs	A dictionary of the outputs of this property.
receptor_residue_name	**Input** - The residue name of the receptor.
required_inputs	The inputs which must be set on this protocol.
restraint_type	**Input** - The type of ligand restraint applied, provided that *apply_restraints* is *True* The default value of this attribute is `RestraintType.Harmonic`.
schema	A serializable schema for this object.
setup_only	**Input** - If true, YANK will only create and validate the setup files, but not actually run any simulations.
solvated_complex_coordinates	**Input** - The file path to the solvated complex coordinates.
solvated_complex_system	**Input** - The parameterized solvated complex system object.
solvated_complex_trajectory_path	**Output** - The file path to the generated ligand trajectory.
solvated_ligand_coordinates	**Input** - The file path to the solvated ligand coordinates.
solvated_ligand_system	**Input** - The parameterized solvated ligand system object.
solvated_ligand_trajectory_path	**Output** - The file path to the generated ligand trajectory.
steps_per_iteration	**Input** - The number of steps per YANK iteration to perform.
thermodynamic_state	**Input** - The state at which to run the calculations.
timestep	**Input** - The length of the timestep to take.
verbose	**Input** - Controls whether or not to run YANK at high verbosity.

class RestraintType(*value*)

The types of ligand restraints available within yank.

ligand_residue_name

Input - The residue name of the ligand. The default value of this attribute is not set and must be set by the user..

Type str

receptor_residue_name

Input - The residue name of the receptor. The default value of this attribute is not set and must be set by the user..

Type str

solvated_ligand_coordinates

Input - The file path to the solvated ligand coordinates. The default value of this attribute is not set and must be set by the user..

Type str

`solvated_ligand_system`
: **Input** - The parameterized solvated ligand system object. The default value of this attribute is not set and must be set by the user..

  Type ParameterizedSystem

`solvated_complex_coordinates`
: **Input** - The file path to the solvated complex coordinates. The default value of this attribute is not set and must be set by the user..

  Type str

`solvated_complex_system`
: **Input** - The parameterized solvated complex system object. The default value of this attribute is not set and must be set by the user..

  Type ParameterizedSystem

`force_field_path`
: **Input** - The path to the force field which defines the charge method to use for the calculation. The default value of this attribute is not set and must be set by the user..

  Type str

`apply_restraints`
: **Input** - Determines whether the ligand should be explicitly restrained to the receptor in order to stop the ligand from temporarily unbinding. The default value of this attribute is `True`.

  Type bool

`restraint_type`
: **Input** - The type of ligand restraint applied, provided that *apply_restraints* is *True* The default value of this attribute is `RestraintType.Harmonic`.

  Type *LigandReceptorYankProtocol.RestraintType*

`ligand_electrostatic_lambdas`
: **Input** - The list of electrostatic alchemical states that YANK should sample at when calculating the free energy of the solvated ligand. If no option is set, YANK will use *trailblaze* algorithm to determine this option automatically. The default value of this attribute is not set. This attribute is *optional*.

  Type list

`ligand_steric_lambdas`
: **Input** - The list of steric alchemical states that YANK should sample at when calculating the free energy of the solvated ligand. If no option is set, YANK will use *trailblaze* algorithm to determine this option automatically. The default value of this attribute is not set. This attribute is *optional*.

  Type list

`complex_electrostatic_lambdas`
: **Input** - The list of electrostatic alchemical states that YANK should sample at when calculating the free energy of the ligand in complex with the receptor. If no option is set, YANK will use *trailblaze* algorithm to determine this option automatically. The default value of this attribute is not set. This attribute is *optional*.

  Type list

`complex_steric_lambdas`
: **Input** - The list of steric alchemical states that YANK should sample at when calculating the free energy of the ligand in complex with the receptor. If no option is set, YANK will use *trailblaze* algorithm to determine this option automatically. The default value of this attribute is not set. This attribute is *optional*.

  Type list

`solvated_ligand_trajectory_path`

Output - The file path to the generated ligand trajectory. The default value of this attribute is not set and must be set by the user..

Type str

`solvated_complex_trajectory_path`

Output - The file path to the generated ligand trajectory. The default value of this attribute is not set and must be set by the user..

Type str

`allow_merging`

Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)

Determines whether this protocol can be merged with another.

Parameters

- **other** (Protocol) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

`checkpoint_interval`

Input - The number of iterations between saving YANK checkpoint files. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is 1.

Type int

`property dependencies`

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

Type list of ProtocolPath

`execute`(*directory=''*, *available_resources=None*)

Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`free_energy_difference`

Output - The estimated free energy difference between the two phases ofinterest. The default value of this attribute is not set and must be set by the user..

Type *Observable*

`classmethod from_json`(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

`classmethod from_schema`(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

`classmethod get_attributes`(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

`get_class_attribute`(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

`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** (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

`gradient_parameters`

Input - An optional list of parameters to differentiate the estimated free energy with respect to.

Type list

`id`

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)

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

Parameters **other** (Protocol) – 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_equilibration_iterations`

Input - The number of iterations used for equilibration before production run. Only post-equilibration iterations are written to file. The default value of this attribute is `1`.

Type int

number_of_iterations
Input - The number of YANK iterations to perform. The default value of this attribute is `5000`.

Type int

property outputs
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

classmethod parse_json(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

setup_only
Input - If true, YANK will only create and validate the setup files, but not actually run any simulations. This argument is mainly only to be used for testing purposes. The default value of this attribute is `False`.

Type bool

steps_per_iteration
Input - The number of steps per YANK iteration to perform. The default value of this attribute is `500`.

Type int

thermodynamic_state
Input - The state at which to run the calculations. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

timestep
Input - The length of the timestep to take. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `2 fs`.

Type Quantity

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

verbose
Input - Controls whether or not to run YANK at high verbosity. The default value of this attribute is `False`.

Type bool

SolvationYankProtocol

class openff.evaluator.protocols.yank.**SolvationYankProtocol**(*protocol_id*)
A protocol for estimating the change in free energy upon transferring a solute into a solvent (referred to as solvent 1) from a second solvent (referred to as solvent 2) by performing an alchemical free energy calculation using the YANK framework.

This protocol can be used for box solvation free energies (setting the *solvent_1* input to the solvent of interest and setting *solvent_2* as an empty *Substance*) or transfer free energies (setting both the *solvent_1* and *solvent_2* inputs to different solvents).

__init__(*protocol_id*)

Methods

`__init__`(protocol_id)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other[, path_replacements])	Determines whether this protocol can be merged with another.
`execute`([directory, available_resources])	Execute the protocol.
`from_json`(file_path)	Create this object from a JSON file.
`from_schema`(schema)	Initializes a protocol from it's schema definition.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.

continues on next page

Table 365 – continued from previous page

`get_class_attribute`(reference_path)	Returns one of this protocols, or any of its children's, attributes directly (rather than its value).
`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.
`json`([file_path, format])	Creates a JSON representation of this class.
`merge`(other)	Merges another Protocol with this one.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Prepend a unique identifier to this protocols id.
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.
`validate`([attribute_type])	Validate the values of the attributes.

Attributes

`allow_merging`	**Input** - Defines whether this protocols is allowed to merge with other protocols.
`checkpoint_interval`	**Input** - The number of iterations between saving YANK checkpoint files.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`electrostatic_lambdas_1`	**Input** - The list of electrostatic alchemical states that YANK should sample at.
`electrostatic_lambdas_2`	**Input** - The list of electrostatic alchemical states that YANK should sample at.
`free_energy_difference`	**Output** - The estimated free energy difference between the solute in thefirst solvent and the second solvent (i.e.
`gradient_parameters`	**Input** - An optional list of parameters to differentiate the estimated free energy with respect to.
`id`	The unique id of this protocol.
`number_of_equilibration_iterations`	**Input** - The number of iterations used for equilibration before production run.
`number_of_iterations`	**Input** - The number of YANK iterations to perform.
`outputs`	A dictionary of the outputs of this property.
`required_inputs`	The inputs which must be set on this protocol.
`schema`	A serializable schema for this object.
`setup_only`	**Input** - If true, YANK will only create and validate the setup files, but not actually run any simulations.
`solute`	**Input** - The substance describing the composition of the solute.
`solution_1_coordinates`	**Input** - The file path to the coordinates of the solute embedded in the first solvent.
`solution_1_free_energy`	**Output** - The free energy change of transforming the an ideal solute molecule into a fully interacting molecule in the first solvent.

continues on next page

Table 366 – continued from previous page

solution_1_system	**Input** - The parameterized system object of the solute embedded in the first solvent.
solution_1_trajectory_path	**Output** - The file path to the trajectory containing the solute in the first solvent.
solution_2_coordinates	**Input** - The file path to the coordinates of the solute embedded in the second solvent.
solution_2_free_energy	**Output** - The free energy change of transforming the an ideal solute molecule into a fully interacting molecule in the second solvent.
solution_2_system	**Input** - The parameterized system object of the solute embedded in the second solvent.
solution_2_trajectory_path	**Output** - The file path to the trajectory containing the solute in the second solvent.
solvent_1	**Input** - The substance describing the composition of the first solvent.
solvent_1_coordinate_path	**Output** - The file path to the coordinates of only the first solvent.
solvent_1_trajectory_path	**Output** - The file path to the trajectory containing only the first solvent.
solvent_2	**Input** - The substance describing the composition of the second solvent.
solvent_2_coordinate_path	**Output** - The file path to the coordinates of only the second solvent.
solvent_2_trajectory_path	**Output** - The file path to the trajectory containing only the second solvent.
steps_per_iteration	**Input** - The number of steps per YANK iteration to perform.
steric_lambdas_1	**Input** - The list of steric alchemical states that YANK should sample at.
steric_lambdas_2	**Input** - The list of steric alchemical states that YANK should sample at.
thermodynamic_state	**Input** - The state at which to run the calculations.
timestep	**Input** - The length of the timestep to take.
verbose	**Input** - Controls whether or not to run YANK at high verbosity.

solute

Input - The substance describing the composition of the solute. This should include the solute molecule as well as any counter ions. The default value of this attribute is not set and must be set by the user..

Type *Substance*

solvent_1

Input - The substance describing the composition of the first solvent. The default value of this attribute is not set and must be set by the user..

Type *Substance*

solvent_2

Input - The substance describing the composition of the second solvent. The default value of this attribute is not set and must be set by the user..

Type *Substance*

solution_1_coordinates

Input - The file path to the coordinates of the solute embedded in the first solvent. The default value of this attribute is not set and must be set by the user..

Type str

`solution_1_system`

Input - The parameterized system object of the solute embedded in the first solvent. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

`solution_2_coordinates`

Input - The file path to the coordinates of the solute embedded in the second solvent. The default value of this attribute is not set and must be set by the user..

Type str

`solution_2_system`

Input - The parameterized system object of the solute embedded in the second solvent. The default value of this attribute is not set and must be set by the user..

Type ParameterizedSystem

`electrostatic_lambdas_1`

Input - The list of electrostatic alchemical states that YANK should sample at. These values will be passed to the YANK *lambda_electrostatics* option. If no option is set, YANK will use *trailblaze* algorithm to determine this option automatically. The default value of this attribute is not set. This attribute is *optional*.

Type list

`steric_lambdas_1`

Input - The list of steric alchemical states that YANK should sample at. These values will be passed to the YANK *lambda_sterics* option. If no option is set, YANK will use *trailblaze* algorithm to determine this option automatically. The default value of this attribute is not set. This attribute is *optional*.

Type list

`electrostatic_lambdas_2`

Input - The list of electrostatic alchemical states that YANK should sample at. These values will be passed to the YANK *lambda_electrostatics* option. If no option is set, YANK will use *trailblaze* algorithm to determine this option automatically. The default value of this attribute is not set. This attribute is *optional*.

Type list

`steric_lambdas_2`

Input - The list of steric alchemical states that YANK should sample at. These values will be passed to the YANK *lambda_sterics* option. If no option is set, YANK will use *trailblaze* algorithm to determine this option automatically. The default value of this attribute is not set. This attribute is *optional*.

Type list

`solution_1_free_energy`

Output - The free energy change of transforming the an ideal solute molecule into a fully interacting molecule in the first solvent. The default value of this attribute is not set and must be set by the user..

Type *Observable*

`solvent_1_coordinate_path`

Output - The file path to the coordinates of only the first solvent. The default value of this attribute is not set and must be set by the user..

Type str

`solvent_1_trajectory_path`
Output - The file path to the trajectory containing only the first solvent. The default value of this attribute is not set and must be set by the user..

Type str

`solution_1_trajectory_path`
Output - The file path to the trajectory containing the solute in the first solvent. The default value of this attribute is not set and must be set by the user..

Type str

`solution_2_free_energy`
Output - The free energy change of transforming the an ideal solute molecule into a fully interacting molecule in the second solvent. The default value of this attribute is not set and must be set by the user..

Type *Observable*

`solvent_2_coordinate_path`
Output - The file path to the coordinates of only the second solvent. The default value of this attribute is not set and must be set by the user..

Type str

`solvent_2_trajectory_path`
Output - The file path to the trajectory containing only the second solvent. The default value of this attribute is not set and must be set by the user..

Type str

`solution_2_trajectory_path`
Output - The file path to the trajectory containing the solute in the second solvent. The default value of this attribute is not set and must be set by the user..

Type str

`free_energy_difference`
Output - The estimated free energy difference between the solute in thefirst solvent and the second solvent (i.e. G = G_1 - G_2). The default value of this attribute is not set and must be set by the user..

Type *Observable*

`allow_merging`
Input - Defines whether this protocols is allowed to merge with other protocols. The default value of this attribute is `True`.

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*, *path_replacements=None*)
Determines whether this protocol can be merged with another.

Parameters

- **other** (`Protocol`) – The protocol to compare against.
- **path_replacements** (*list of tuple of str, optional*) – Replacements to make in any value reference protocol paths before comparing for equality.

Returns True if the two protocols are safe to merge.

Return type bool

`checkpoint_interval`
Input - The number of iterations between saving YANK checkpoint files. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is 1.

Type int

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

Type list of ProtocolPath

`execute`(*directory=''*, *available_resources=None*)
Execute the protocol.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on. If *None*, the protocol will be executed on a single CPU.

`classmethod from_json`(*file_path*)
Create this object from a JSON file.

Parameters **file_path** (*str*) – The path to load the JSON from.

Returns The parsed class.

Return type cls

classmethod from_schema(*schema*)

Initializes a protocol from it's schema definition.

Parameters **schema** (ProtocolSchema) – The schema to initialize the protocol using.

Returns The initialized protocol.

Return type cls

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

get_class_attribute(*reference_path*)

Returns one of this protocols, or any of its children's, attributes directly (rather than its value).

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

Returns The class attribute.

Return type object

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** (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

gradient_parameters

Input - An optional list of parameters to differentiate the estimated free energy with respect to.

Type list

id

The unique id of this protocol. The default value of this attribute is not set and must be set by the user..

Type str

`json`(*file_path=None*, *format=False*)
Creates a JSON representation of this class.

Parameters

- **file_path** (*str, optional*) – The (optional) file path to save the JSON file to.
- **format** (*bool*) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

`merge`(*other*)
Merges another Protocol with this one. The id of this protocol will remain unchanged.

Parameters **other** (`Protocol`) – 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_equilibration_iterations`
Input - The number of iterations used for equilibration before production run. Only post-equilibration iterations are written to file. The default value of this attribute is `1`.

Type int

`number_of_iterations`
Input - The number of YANK iterations to perform. The default value of this attribute is `5000`.

Type int

`property outputs`
A dictionary of the outputs of this property.

Type dict of ProtocolPath and Any

`classmethod parse_json`(*string_contents*)
Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (*str or bytes*) – The typed json string.

Returns The parsed class.

Return type Any

`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 required_inputs
The inputs which must be set on this protocol.

Type list of ProtocolPath

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Prepend a unique identifier to this protocols id. If the id already has a prepended uuid, it will be overwritten by this value.

Parameters **value** (*str*) – The uuid to prepend.

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.

setup_only
Input - If true, YANK will only create and validate the setup files, but not actually run any simulations. This argument is mainly only to be used for testing purposes. The default value of this attribute is `False`.

Type bool

steps_per_iteration
Input - The number of steps per YANK iteration to perform. The default value of this attribute is `500`.

Type int

thermodynamic_state
Input - The state at which to run the calculations. The default value of this attribute is not set and must be set by the user..

Type *ThermodynamicState*

timestep
Input - The length of the timestep to take. When two protocols are merged, the largest value of this attribute from either protocol is retained. The default value of this attribute is `2 fs`.

Type Quantity

validate(*attribute_type=None*)
Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (*type of Attribute, optional*) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

verbose
Input - Controls whether or not to run YANK at high verbosity. The default value of this attribute is `False`.

Type bool

2.32.11 Workflow Construction Utilities

SimulationProtocols	The common set of protocols which would be required to estimate an observable by running a new molecule simulation.
ReweightingProtocols	The common set of protocols which would be required to re-weight an observable from cached simulation data.
generate_base_reweighting_protocols	Constructs a set of protocols which, when combined in a workflow schema, may be executed to reweight a set of cached simulation data to estimate the average value of an observable.
generate_reweighting_protocols	
generate_simulation_protocols	Constructs a set of protocols which, when combined in a workflow schema, may be executed to run a single simulation to estimate the average value of an observable.

SimulationProtocols

class openff.evaluator.protocols.utils.**SimulationProtocols**(*build_coordinates:* openff.evaluator.protocols.coordinates.BuildCoordinatesPa *assign_parameters:* openff.evaluator.protocols.forcefield.BaseBuildSystem, *energy_minimisation:* openff.evaluator.protocols.openmm.OpenMMEnergyMini *equilibration_simulation:* openff.evaluator.protocols.openmm.OpenMMSimulation, *production_simulation:* openff.evaluator.protocols.openmm.OpenMMSimulation, *analysis_protocol:* *openff.evaluator.protocols.utils.S*, *converge_uncertainty:* openff.evaluator.workflow.protocols.ProtocolGroup, *decorrelate_trajectory:* openff.evaluator.protocols.analysis.DecorrelateTrajectory, *decorrelate_observables:* openff.evaluator.protocols.analysis.DecorrelateObservable

The common set of protocols which would be required to estimate an observable by running a new molecule simulation.

__init__(*build_coordinates:* openff.evaluator.protocols.coordinates.BuildCoordinatesPackmol, *assign_parameters:* openff.evaluator.protocols.forcefield.BaseBuildSystem, *energy_minimisation:* openff.evaluator.protocols.openmm.OpenMMEnergyMinimisation, *equilibration_simulation:* openff.evaluator.protocols.openmm.OpenMMSimulation, *production_simulation:* openff.evaluator.protocols.openmm.OpenMMSimulation, *analysis_protocol:* *openff.evaluator.protocols.utils.S*, *converge_uncertainty:* openff.evaluator.workflow.protocols.ProtocolGroup, *decorrelate_trajectory:* openff.evaluator.protocols.analysis.DecorrelateTrajectory, *decorrelate_observables:* openff.evaluator.protocols.analysis.DecorrelateObservables) → None

Methods

`__init__`(build_coordinates, ...)

Attributes

`build_coordinates`
`assign_parameters`
`energy_minimisation`
`equilibration_simulation`
`production_simulation`
`analysis_protocol`
`converge_uncertainty`
`decorrelate_trajectory`
`decorrelate_observables`

ReweightingProtocols

class openff.evaluator.protocols.utils.**ReweightingProtocols**(*unpack_stored_data:* openff.evaluator.protocols.storage.UnpackStoredSimulati
join_trajectories: openff.evaluator.protocols.reweighting.ConcatenateTraje
join_observables: openff.evaluator.protocols.reweighting.ConcatenateObser
build_reference_system: openff.evaluator.protocols.forcefield.BaseBuildSystem,
evaluate_reference_potential: openff.evaluator.protocols.reweighting.BaseEvaluateEner
build_target_system: openff.evaluator.protocols.forcefield.BaseBuildSystem,
evaluate_target_potential: openff.evaluator.protocols.reweighting.BaseEvaluateEner
statistical_inefficiency: *openff.evaluator.protocols.utils.S*,
replicate_statistics: openff.evaluator.protocols.miscellaneous.DummyProtoco
decorrelate_reference_potential: openff.evaluator.protocols.analysis.DecorrelateObservab
decorrelate_target_potential: openff.evaluator.protocols.analysis.DecorrelateObservab
decorrelate_observable: openff.evaluator.protocols.analysis.DecorrelateObservab
*zero_gradients: Optional[*openff.evaluator.protocols.gradients.ZeroGradient
reweight_observable: *openff.evaluator.protocols.utils.T*)

The common set of protocols which would be required to re-weight an observable from cached simulation data.

__init__(*unpack_stored_data:* openff.evaluator.protocols.storage.UnpackStoredSimulationData, *join_trajectories:* openff.evaluator.protocols.reweighting.ConcatenateTrajectories, *join_observables:* openff.evaluator.protocols.reweighting.ConcatenateObservables, *build_reference_system:* openff.evaluator.protocols.forcefield.BaseBuildSystem, *evaluate_reference_potential:* openff.evaluator.protocols.reweighting.BaseEvaluateEnergies, *build_target_system:* openff.evaluator.protocols.forcefield.BaseBuildSystem, *evaluate_target_potential:* openff.evaluator.protocols.reweighting.BaseEvaluateEnergies, *statistical_inefficiency: openff.evaluator.protocols.utils.S*, *replicate_statistics:* openff.evaluator.protocols.miscellaneous.DummyProtocol, *decorrelate_reference_potential:* openff.evaluator.protocols.analysis.DecorrelateObservables, *decorrelate_target_potential:* openff.evaluator.protocols.analysis.DecorrelateObservables, *decorrelate_observable:* openff.evaluator.protocols.analysis.DecorrelateObservables, *zero_gradients: Optional[*openff.evaluator.protocols.gradients.ZeroGradients*]*, *reweight_observable: openff.evaluator.protocols.utils.T*) → None

Methods

__init__(unpack_stored_data, ...)

Attributes

`unpack_stored_data`
`join_trajectories`
`join_observables`
`build_reference_system`
`evaluate_reference_potential`
`build_target_system`
`evaluate_target_potential`
`statistical_inefficiency`
`replicate_statistics`
`decorrelate_reference_potential`
`decorrelate_target_potential`
`decorrelate_observable`
`zero_gradients`
`reweight_observable`

generate_base_reweighting_protocols

`openff.evaluator.protocols.utils.`**`generate_base_reweighting_protocols`**`(`*statistical_inefficiency: openff.evaluator.protocols.utils.S*, *reweight_observable: openff.evaluator.protocols.utils.T*, *replicator_id: str = 'data_replicator'*, *id_suffix: str = ''*) → Tuple[*openff.evaluator.protocols.utils.Reweight* openff.evaluator.protocols.utils.T], *openff.evaluator.workflow.schemas.ProtocolR*

Constructs a set of protocols which, when combined in a workflow schema, may be executed to reweight a set of cached simulation data to estimate the average value of an observable.

Parameters

- **statistical_inefficiency** – The protocol which will be used to compute the statistical inefficiency and equilibration time of the observable of interest. This information will be used to decorrelate the cached data prior to reweighting.
- **reweight_observable** – The MBAR reweighting protocol to use to reweight the observable to the target state. This method will automatically set the reduced potentials on the object.
- **replicator_id** (*str*) – The id to use for the cached data replicator.
- **id_suffix** (*str*) – A string suffix to append to each of the protocol ids.

Returns

- *The protocols to add to the workflow, a reference to the average value of the*
- estimated observable (an `Observable` object), and the replicator which will
- *clone the workflow for each piece of cached simulation data.*

generate_reweighting_protocols

`openff.evaluator.protocols.utils.`**`generate_reweighting_protocols`**(*observable_type:* openff.evaluator.utils.observables.ObservableType, *replicator_id:* *str* = *'data_replicator'*, *id_suffix:* *str* = *''*) → Tuple[*openff.evaluator.protocols.utils.ReweightingPro* *openff.evaluator.protocols.reweighting.ReweightOb* *openff.evaluator.workflow.schemas.ProtocolReplica*

generate_simulation_protocols

`openff.evaluator.protocols.utils.`**`generate_simulation_protocols`**(*analysis_protocol: openff.evaluator.protocols.utils.S*, *use_target_uncertainty:* *bool*, *id_suffix:* *str* = *''*, *conditional_group: Optional[*openff.evaluator.protocols.groups.Conditional*= None*, *n_molecules:* *int* = *1000*) → Tuple[*openff.evaluator.protocols.utils.SimulationProto* *openff.evaluator.workflow.utils.ProtocolPath*, *openff.evaluator.storage.data.StoredSimulationData*]

Constructs a set of protocols which, when combined in a workflow schema, may be executed to run a single simulation to estimate the average value of an observable.

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`** – The protocol which will extract the observable of interest from the generated simulation data.
- **`use_target_uncertainty`** – Whether to run the simulation until the observable is estimated to within the target uncertainty.
- **`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.
- **`n_molecules`** (*`int`*) – The number of molecules to use in the workflow.

Returns

- *The protocols to add to the workflow, a reference to the average value of the*
- estimated observable (an `Observable` object), and an object which describes
- *the default data from a simulation to store, such as the uncorrelated statistics*
- *and configurations.*

2.32.12 Attribute Utilities

`Attribute`	A custom descriptor used to add useful metadata to class attributes.
`AttributeClass`	A base class for objects which require well defined attributes with additional metadata.
`UNDEFINED`	A custom type used to differentiate between `None` values, and an undeclared optional value.
`PlaceholderValue`	A class to act as a place holder for an attribute whose value is not known a priori, but will be set later by some specialised code.

Attribute

class `openff.evaluator.attributes.`**`Attribute`**(*docstring*, *type_hint*, *default_value=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *optional=False*, *read_only=False*)

A custom descriptor used to add useful metadata to class attributes.

This decorator expects the object to have a matching private field in addition to the public attribute. For example if an object has an attribute *substance*, the object must also have a *_substance* field.

Notes

The attribute class will automatically create this private attribute on the object and populate it with the default value.

`__init__`(*docstring*, *type_hint*, *default_value=<openff.evaluator.attributes.attributes.UndefinedAttribute object>*, *optional=False*, *read_only=False*)

Initializes a new Attribute object.

Parameters

- **`docstring`** (*`str`*) – A docstring describing the attributes purpose. This will automatically be decorated with additional information such as type hints, default values, etc.
- **`type_hint`** (*`type, typing.Union`*) – The expected type of this attribute. This will be used to help the workflow engine ensure that expected input types match corresponding output values.
- **`default_value`** (*`Any`*) – The default value for this attribute.
- **`optional`** (*`bool`*) – Defines whether this is an optional input of a class. If true, the *default_value* should be set to *UNDEFINED*.
- **`read_only`** (*`bool`*) – Defines whether this attribute is read-only.

Methods

`__init__`(docstring, type_hint[, ...])	Initializes a new Attribute object.

AttributeClass

class `openff.evaluator.attributes.`**`AttributeClass`**

A base class for objects which require well defined attributes with additional metadata.

`__init__`()

Methods

`__init__`()	
`from_json`(file_path)	Create this object from a JSON file.
`get_attributes`([attribute_type])	Returns all attributes of a specific *attribute_type*.
`json`([file_path, format])	Creates a JSON representation of this class.
`parse_json`(string_contents)	Parses a typed json string into the corresponding class structure.
`validate`([attribute_type])	Validate the values of the attributes.

validate(*attribute_type=None*)

Validate the values of the attributes. If *attribute_type* is set, only attributes of that type will be validated.

Parameters **attribute_type** (`type of Attribute, optional`) – The type of attribute to validate.

Raises `ValueError` **or** `AssertionError` –

classmethod get_attributes(*attribute_type=None*)

Returns all attributes of a specific *attribute_type*.

Parameters **attribute_type** (`type of Attribute, optional`) – The type of attribute to search for.

Returns The names of the attributes of the specified type.

Return type list of str

classmethod parse_json(*string_contents*)

Parses a typed json string into the corresponding class structure.

Parameters **string_contents** (`str or bytes`) – The typed json string.

Returns The parsed class.

Return type Any

classmethod from_json(*file_path*)

Create this object from a JSON file.

Parameters **file_path** (`str`) – The path to load the JSON from.

Returns The parsed class.

Return type cls

json(*file_path=None*, *format=False*)

Creates a JSON representation of this class.

Parameters

- **file_path** (`str, optional`) – The (optional) file path to save the JSON file to.
- **format** (`bool`) – Whether to format the JSON or not.

Returns The JSON representation of this class.

Return type str

UNDEFINED

openff.evaluator.attributes.**UNDEFINED** = **<openff.evaluator.attributes.attributes.UndefinedAttribute object>**

A custom type used to differentiate between `None` values, and an undeclared optional value.

PlaceholderValue

class openff.evaluator.attributes.**PlaceholderValue**

A class to act as a place holder for an attribute whose value is not known a priori, but will be set later by some specialised code. This may include the input to a protocol which will be set by a workflow as the output of an executed protocol.

__init__()

Methods

__init__()	

2.32.13 Observable Utilities

Observable	A class which stores the mean value of an observable as well as the standard error in the mean.
ObservableArray	A class which stores the value(s) of an observable obtained via molecule simulation (or simulation data) as well as optionally the derivatives of the value with respect to certain force field parameters.
ObservableType	An enumeration of the common observables which may be extracted from molecular simulations (or simulation data) and stored in an `ObservableFrame`.
ObservableFrame	A data object for storing and retrieving frames of the thermodynamic observables enumerated by the `ObservableType` enum.
bootstrap	Bootstrapping a set of observables to compute the average value of the observables as well as the the standard error in the average.

Observable

class openff.evaluator.utils.observables.**Observable**(*value: Optional[Union[openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity]] = None*, *gradients: Optional[List[*openff.evaluator.forcefield.gradients.ParameterGradient *= None*)

A class which stores the mean value of an observable as well as the standard error in the mean. Optionally, the derivatives of the mean with respect to certain force field parameters may also be stored.

__init__(*value: Optional[Union[openff.evaluator.utils.units.Measurement, openff.evaluator.utils.units.Quantity]] = None*, *gradients: Optional[List[*openff.evaluator.forcefield.gradients.ParameterGradient*]] = None*)

Methods

`__init__`([value, gradients])	
`clear_gradients`()	Clears all gradient information.

Attributes

`error`
`gradients`
`value`

clear_gradients()

Clears all gradient information.

ObservableArray

class openff.evaluator.utils.observables.**ObservableArray**(*value: Optional[openff.evaluator.utils.units.Quantity] = None*, *gradients: Optional[List[*openff.evaluator.forcefield.gradients.ParameterGr

= None)

A class which stores the value(s) of an observable obtained via molecule simulation (or simulation data) as well as optionally the derivatives of the value with respect to certain force field parameters.

__init__(*value: Optional[openff.evaluator.utils.units.Quantity] = None*, *gradients: Optional[List[*openff.evaluator.forcefield.gradients.ParameterGradient*]] = None*)

Methods

`__init__`([value, gradients])	
`clear_gradients`()	Clears all gradient information.
`join`(*observables)	Concatenates multiple observables together in the order that they appear in the args list.
`subset`(indices)	Extracts the subset of the values stored for this observable at the specified indices.

Attributes

`gradients`	
value	The value(s) of the observable.

property value: openff.evaluator.utils.units.Quantity
The value(s) of the observable.

subset(*indices: Iterable[int]*) → *openff.evaluator.utils.observables.ObservableArray*
Extracts the subset of the values stored for this observable at the specified indices.

Parameters **indices** – The indices of the entries to extract.

Returns

Return type The subset of the observable values.

classmethod join(**observables:* openff.evaluator.utils.observables.ObservableArray) → *openff.evaluator.utils.observables.ObservableArray*
Concatenates multiple observables together in the order that they appear in the args list.

Parameters **observables** – The observables to join.

Returns

Return type The concatenated observable object.

clear_gradients()
Clears all gradient information.

ObservableType

class openff.evaluator.utils.observables.**ObservableType**(*value*)
An enumeration of the common observables which may be extracted from molecular simulations (or simulation data) and stored in an `ObservableFrame`.

__init__()

Attributes

`PotentialEnergy`
`KineticEnergy`
`TotalEnergy`
`Temperature`
`Volume`
`Density`

continues on next page

Table 381 – continued from previous page

Enthalpy	
ReducedPotential	

ObservableFrame

class openff.evaluator.utils.observables.**ObservableFrame**(*observables: Optional[Dict[Union[str, openff.evaluator.utils.observables.ObservableType], openff.evaluator.utils.observables.ObservableArray]] = None*)

A data object for storing and retrieving frames of the thermodynamic observables enumerated by the `ObservableType` enum.

__init__(*observables: Optional[Dict[Union[str, openff.evaluator.utils.observables.ObservableType], openff.evaluator.utils.observables.ObservableArray]] = None*)

Methods

__init__([observables])	
clear()	
clear_gradients()	Clears all gradient information for each observable in the frame.
from_openmm(file_path[, pressure])	Creates an observable frame from the CSV output of an OpenMM simulation.
get(k[,d])	
items()	
join(*observable_frames)	Joins multiple observable frames together in the order that they appear in the args list.
keys()	
pop(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised.
popitem()	as a 2-tuple; but raise KeyError if D is empty.
setdefault(k[,d])	
subset(indices)	Extracts the subset of the the array which is located at the specified indices.
update([E,]**F)	If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v
values()	

`classmethod from_openmm`(*file_path: str*, *pressure: Optional[openff.evaluator.utils.units.Quantity] = None*) → *openff.evaluator.utils.observables.ObservableFrame*

Creates an observable frame from the CSV output of an OpenMM simulation.

Parameters

- **`file_path`** – The file path to the CSV file.
- **`pressure`** – The pressure at which the observables in the csv file were collected.

Returns

Return type The imported observables.

`subset`(*indices: Iterable[int]*) → *openff.evaluator.utils.observables.ObservableFrame*

Extracts the subset of the the array which is located at the specified indices.

Parameters **`indices`** – The indices of the entries to extract.

Returns

Return type The subset of data.

`classmethod join`(**observable_frames: openff.evaluator.utils.observables.ObservableFrame*) → *openff.evaluator.utils.observables.ObservableFrame*

Joins multiple observable frames together in the order that they appear in the args list.

Parameters **`observable_frames`** – The observable frames to join.

Returns

Return type The joined observable frame.

`clear_gradients`()

Clears all gradient information for each observable in the frame.

`clear`() → None. Remove all items from D.

`get`(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

`items`() → a set-like object providing a view on D's items

`keys`() → a set-like object providing a view on D's keys

`pop`(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

`popitem`() → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise KeyError if D is empty.

`setdefault`(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

`update`([*E*], ***F*) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

`values`() → an object providing a view on D's values

bootstrap

openff.evaluator.utils.observables.**bootstrap**(*bootstrap_function: Callable*, *iterations: int = 200*, *relative_sample_size: float = 1.0*, *sub_counts: Optional[Iterable[int]] = None*, ***observables: openff.evaluator.utils.observables.ObservableArray*) → *openff.evaluator.utils.observables.Observable*

Bootstrapping a set of observables to compute the average value of the observables as well as the the standard error in the average.

Parameters

- **bootstrap_function** – The function to evaluate at each bootstrap iteration.
- **iterations** – The number of bootstrap iterations to perform.
- **relative_sample_size** – The percentage sample size to bootstrap over, relative to the size of the full data set.
- **sub_counts** – If the data being bootstrapped contains arrays of concatenated sub data (such as when reweighting), this variable can be used to specify the number of items which belong to each subset. Data is then sampled with replacement so that the bootstrap sample contains the correct proportion of data from each subset.

  If the data to bootstrap is of the form [x0, x1, x2, y0, y1] for example, then *data_sub_counts=[3, 2]* and a possible sample may look like [x0, x0, x2, y0, y0], but never [x0, x1, y0, y1, y1].

  The sub-counts must sum up to the total length of the data provided to `observables`.
- **observables** – The observables which will be passed to the bootstrap function. All observables must have the same length.

Returns

Return type The average of the data and the uncertainty in the average.

2.32.14 Plug-in Utilities

Plug-ins

register_default_plugins	Registers the built-in workflow protocols, calculation layers and physical properties with the plugin system.
register_external_plugins	Registers any supported plugins found in external packages with the plugin system.

register_default_plugins

`openff.evaluator.plugins.register_default_plugins()`

Registers the built-in workflow protocols, calculation layers and physical properties with the plugin system.

register_external_plugins

`openff.evaluator.plugins.register_external_plugins()`

Registers any supported plugins found in external packages with the plugin system.

2.33 Release History

Releases 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

2.33.1 0.3.9

Bugfixes

- PR #402: Fix importing full ThermoML archive

Behaviour Changes

The way that ThermoML archive files are served was changed in 2021 so that individual journal archives are no longer made available. Instead, now only the full ThermoML archive can be downloaded. Because of this, the `ImportThermoMLDataSchema` schema no longer allows users to select which journal to pull data from.

2.33.2 0.3.8

Bugfixes

- PR #390: Fix excluding v-sites from OpenMM positions

2.33.3 0.3.7

Bugfixes

- PR #389: Fix v-site positions not set by OpenMM

2.33.4 0.3.6

Bugfixes

- PR #375: Fix #374 - import from collections.abc
- PR #379: Fix #378 - ‘FilterDuplicates` unintentionally selects values without uncertainty if multiple are present
- PR #384: Fix #382 - Default keyword arguments result in error
- PR #387: Fix #380 - Recursion error in local file storage

New Features

- PR #385: Support custom OpenMM nonbonded forces
- PR #386: Migrate to new OpenMM namespace

2.33.5 0.3.5

Bugfixes

- PR #367: Fix #365 - to/from_pandas does not roundtrip.
- PR #368: Fix #364 - Parsing an invalid IUPAC name raises an exception rather than a warning.
- PR #371: Fix gradients of non-Quantity parameters.

New Features

- PR #362: Support dask-jobqueue Slurm backend.
- PR #366: Support gradients of handler attributes.

2.33.6 0.3.4

A patch release which adds the option (and enables it by default) to remove working files, such as simulated trajectories, when they are no longer needed.

Behaviour Changes

- PR #349: Working files are deleted by default after an estimation batch completes.

2.33.7 0.3.3

This release facilitates the migration of the *openff-evaluator* package from *omnia* to *conda-forge*. This mainly involves changes which update the package to use the new namespaces introduced in the *openff-tookit* package, rather than the old and now deprecated *openforcefield* namespaces.

Bugfixes

- PR #346: Remove the unsupported *encoding* json kwarg.

New Features

- PR #341: Replace usages of dynamic Pint classes with internal static variants.
- PR #343: Migrate to the new OpenFF Toolkit namespace.
- PR #345: Migrate all reference from *omnia* to *conda-forge*.

2.33.8 0.3.2

This release exposes the option to disable caching of simulation data by an evaluator server. The performance of the local storage backend is currently poor when dealing with large amounts of cached data and hence it may be preferable to disable caching in such cases.

New Features

- PR #337: Expose server option to dis/enable data caching.

2.33.9 0.3.1

This release fixes a bug introduced in version 0.3.0 of this framework, whereby the default workflows for computing excess properties could in rare cases be incorrectly merged leading to downstream protocols taking their inputs from the wrong upstream protocol outputs.

While this bug should not affect most calculations, it is recommended that any production calculations performed using version 0.3.0 of this framework be repeated using version 0.3.1.

Bugfixes

- PR #331: Fixes merging excess properties.

2.33.10 0.3.0

The main feature of this release is the overhauling of how the framework computes the gradients of observables with respect to force field parameters.

In particular, from this release onwards all gradients will be computed using the fluctuation formula (also referred to as the thermodynamic gradient), rather than calculation be the re-weighted finite difference approach (PR #280). In general the two methods produce gradients which are numerically indistinguishable, and so this should not markedly change any scientific output of this framework.

The change was made to, in future, enable better integration with automatic differentiation libraries such as jax, and differentiable simulation engines such as timemachine which readily and rapidly give access to $\mathrm{d}U/\mathrm{d}\theta_i$.

Additionally, as of version 0.3.0 'known' charges (i.e. those assigned to TIP3P water and ions) are no longer automatically applied when using a SMIRNOFF based force field. This feature was originally included in the framework as the OpenFF toolkit did not support defining charges on specific molecules in the force field itself. This is now fully supported through the `LibraryCharges` section of a SMIRNOFF force field and hence this workaround is no longer required. From now on all ion and water charges **must** be specified in the SMIRNOFF force field.

Finally, this release includes **beta** support for computing host-guest binding affinities using the attach-pull-release (APR) method through integration with the pAPRika and taproom packages. This support was largely facilitated by the efforts of the `paprika` authors - David R. Slochower and Jeffry Setiadi.

Bugfixes

- PR #285: Use merged protocols in workflow provenance.
- PR #287: Fix merging of nested protocol inputs

New Features

- PR #262: Initial host-guest binding affinity support via `paprika` and `taproom`.
- PR #280: Switch to computing thermodynamic gradients.
- PR #309: Add a date to the timestamp logging output.
- PR #311: Initial solvation free energy gradient support.
- PR #312: Support caching free energy data.
- PR #324: Adds new miscellaneous `DummyProtocol` protocol.

Behaviour Changes

- PR #280: Migrate to thermodynamic gradients.
- PR #310: The SMIRNOFF protocol no longer applies 'known' charges (i.e. water and ions).
- PR #316: Add library charges to the TIP3P test data file.
- PR #328: Store workflow provenance as serialized string.

Breaking Changes

- The `StatisticsArray` array has been completely removed and replaced with a new set of observable (`Observable`, `ObservableArray`, `ObservableFrame` objects (#279, #286).
- The following protocol inputs / outputs have been renamed:
  - `SolvationYankProtocol.solvent_X_system` -> `SolvationYankProtocol.solution_X_system`
  - `SolvationYankProtocol.solvent_X_coordinates` -> `SolvationYankProtocol.solution_X_coordinates`
  - `SolvationYankProtocol.estimated_free_energy` -> `SolvationYankProtocol.free_energy_difference`
- The following classes have been renamed:
  - `OpenMMReducedPotentials` -> `OpenMMEvaluateEnergies`.
  - `AveragePropertyProtocol` -> `BaseAverageObservable`, `ExtractAverageStatistic` -> `AverageObservable`, `ExtractUncorrelatedData` -> `BaseDecorrelateProtocol`, `ExtractUncorrelatedTrajectoryData` -> `DecorrelateTrajectory`, `ExtractUncorrelatedStatisticsData` -> `DecorrelateObservables`
  - `ConcatenateStatistics` -> `ConcatenateObservables`, `BaseReducedPotentials` -> `BaseEvaluateEnergies`, `ReweightStatistics -> ReweightObservable`
- The following classes have been removed:
  - `OpenMMGradientPotentials`, `BaseGradientPotentials`, `CentralDifferenceGradient`
- The final value estimated by a workflow must now be an `Observable` object which contains any gradient information to return. (#296).

2.33.11 0.2.2

This release adds documentation for how physical properties are computed within the framework (both for this, and for previous releases.

Documentation

- PR #281: Initial pass at physical property documentation.

2.33.12 0.2.1

A patch release offering minor bug fixes and quality of life improvements.

Bugfixes

- PR #259: Adds `is_file_and_not_empty` and addresses OpenMM failure modes.
- PR #275: Workaround for N substance molecules > user specified maximum.

New Features

- PR #267: Adds workflow protocol to Boltzmann average free energies.
- PR #269: Expose exclude exact amount from max molecule cap.

2.33.13 0.2.0

This release overhauls the frameworks data curation abilities. In particular, it adds

- a significant amount of data filters, including to filter by state, substance composition and chemical functionalities.

and components to

- easily import all of the ThermoML and FreeSolv archives.
- convert between property types (currently density <-> excess molar volume).
- select data points close to a set of target states, and substances which contain specific functionalities (i.e. select only data points measured for ketones, alcohols or alkanes).

More information about the new curation abilities can be found *in the documentation here*.

New Features

- PR #260: Data set curation overhaul.
- PR #261: Adds `PhysicalPropertyDataSet.from_pandas`.

Breaking Changes

- All of the `PhysicalPropertyDataSet.filter_by_XXX` functions have now been removed in favor of the new curation components. See the *documentation* for information about the newly available filters and more.

2.33.14 0.1.2

A patch release offering minor bug fixes and quality of life improvements.

Bugfixes

- PR #254: Fix incompatible protocols being merged due to an id replacement bug.
- PR #255: Fix recursive `ThermodynamicState` string representation.
- PR #256: Fix incorrect version when installing from tarballs.

2.33.15 0.1.1

A patch release offering minor bug fixes and quality of life improvements.

Bugfixes

- PR #249: Fix replacing protocols of non-existent workflow schema.
- PR #253: Fix *antechamber* truncating charge file.

Documentation

- PR #252: Use *conda-forge* for *ambertools* installation.

2.33.16 0.1.0 - OpenFF Evaluator

Introducing the OpenFF Evaluator! The release marks a significant milestone in the development of this project, and constitutes an almost full redesign of the framework with a focus on stability and ease of use.

Note: *because of the extensive changes made throughout the entire framework, this release should almost be considered as an entirely new package. No files produced by previous versions of this will work with this new release.*

Clearer Branding

First and foremost, this release marks the complete rebranding from the previously named *propertyestimator* to the new *openff-evaluator* package. This change is accompanied by the introduction of a new `openff` namespace for the package, signifying it's position in the larger Open Force Field infrastructure and piplelines.

What was previously:

```
import propertyestimator
```

now becomes:

```
import openff.evaluator
```

The rebranded package is now shipped on `conda` under the new name of `openff-evaluator`:

```
conda install -c conda-forge -c omnia openff-evaluator
```

Markedly Improved Documentation

In addition, the release includes for the first time a significant amount of documentation for using the **`framework and it's features`_** as well as a collection of user focused tutorials which can be ran directly in the browser.

Support for RDKit

This release almost entirely removes the dependence on OpenEye thanks to support for RDKit almost universally across the framework.

The only remaining instance where OpenEye is still required is for host-guest binding affinity calculations where it is used to perform docking.

Model Validation

Starting with this release almost all models, range from `PhysicalProperty` entries to `ProtocolSchema` objects, are now heavily validated to help catch any typos or errors early on.

Batching of Similar Properties

The `EvaluatorServer` now more intelligently attempts to batch properties which may be computed using the same simulations into a single batch to be estimated. While the behaviour was already supported for pure properties in previous, this has now been significantly expanded to work well with mixture properties.

2.33.17 0.0.9 - Multi-state Reweighting Fix

This release implements a fix for calculating the gradients of properties being estimated by reweighting data cached from multiple independant simulations.

Bugfixes

- PR #143: Fix for multi-state gradient calculations.

2.33.18 0.0.8 - ThermoML Improvements

This release is centered around cleaning up the ThermoML data set utilities. The main change is that ThermoML archive files can now be loaded even if they don't contain measurement uncertainties.

New Features

- PR #142: ThermoML archives without uncertainties can now be loaded.

Breaking Changes

- PR #142: All *ThermoMLXXX* classes other than *ThermoMLDataSet* are now private.

2.33.19 0.0.7 - Bug Quick Fixes

This release aims to fix a number of minor bugs.

Bugfixes

- PR #136: Fix for comparing thermodynamic states with unset pressures.
- PR #138: Fix for a typo in the maximum number of minimization iterations.

2.33.20 0.0.6 - Solvation Free Energies

This release centers around two key changes -

i) a general refactoring of the protocol classes to be much cleaner and extensible through the removal of the old stub functions and the addition of cleaner descriptors.

ii) the addition of workflows to estimate solvation free energies via the new `SolvationYankProtocol` and `SolvationFreeEnergy` classes.

The implemented free energy workflow is still rather basic, and does not yet support calculating parameter gradients or estimation from cached simulation data through reweighting.

A new table has been added to the documentation to make clear which built-in properties support which features.

New Features

- PR #110: Cleanup and refactor of protocol classes.
- PR #125: Support for PBS based HPC clusters.
- PR #127: Adds a basic workflow for estimating solvation free energies with YANK.
- PR #130: Adds a cleaner mechanism for restarting simulations from checkpoints.
- PR #134: Update to a more stable dask version.

Bugfixes

- PR #128: Removed the defunct dask backend *processes* kwarg.
- PR #133: Fix for tests failing on MacOS due to *travis* issues.

Breaking Changes

- PR #130: The `RunOpenMMSimulation.steps` input has now been split into the `steps_per_iteration` and `total_number_of_iterations` inputs.

Migration Guide

This release contained several public API breaking changes. For the most part, these can be remedied by the follow steps:

- Replace all instances of `run_openmm_simulation_protocol.steps` to `run_openmm_simulation_protocol.steps_per_iteration`

2.33.21 0.0.5 - Fix For Merging of Estimation Requests

This release implements a fix for a major bug which caused incorrect results to be returned when submitting multiple estimation requests at the same time - namely, the returned results became jumbled between the different requests. As an example, if a request was made to estimate a data set using the *smirnoff99frosst* force field, and then straight after with the *gaff 1.81* force field, the results of the *smirnoff99frosst* request may contain some properties estimated with *gaff 1.81* and vice versa.

This issue does not affect cases where only a single request was made and completed at a time (i.e the results of the previous request completed before the next estimation request was made).

Bugfixes

- PR #119: Fixes gather task merging.
- PR #121: Update to distributed 2.5.1.

2.33.22 0.0.4 - Initial Support for Non-SMIRNOFF FFs

This release adds initial support for estimating property data sets using force fields not based on the `SMIRNOFF` specification. In particular, initial AMBER force field support has been added, along with a protocol which applies said force fields using `tleap`.

New Features

- PR #96: Adds a mechanism for specifying force fields not in the `SMIRNOFF` spec.
- PR #99: Adds support for applying `AMBER` force field parameters through `tleap`
- PR #111: Protocols now stream trajectories from disk, rather than pre-load the whole thing.
- PR #112: Specific types of protocols can now be easily be replaced using `WorkflowOptions`.
- PR #117: Adds support for converting `PhysicalPropertyDataSet` objects to `pandas.DataFrame`.

Bugfixes

- PR #115: Fixes caching data for substances whose smiles contain forward slashes.
- PR #116: Fixes inconsistent mole fraction rounding.

Breaking Changes

- PR #96: The `PropertyEstimatorClient.request_estimate(force_field=...` argument has been renamed to `force_field_source`.

Migration Guide

This release contained several public API breaking changes. For the most part, these can be remedied by the follow steps:

- Change all instances of `PropertyEstimatorClient.request_estimate(force_field=...)` to `PropertyEstimatorClient.request_estimate(force_field_source=...)`

2.33.23 0.0.3 - ExcessMolarVolume and Typing Improvements

This release implements a number of bug fixes and adds two key new features, namely built in support for estimating excess molar volume measurements, and improved type checking for protocol inputs and outputs.

New Features

- PR #98: `Substance` objects may now have components with multiple amount types.
- PR #101: Added support for estimating `ExcessMolarVolume` measurements from simulations.
- PR #104: `typing.Union` is now a valid type arguemt to `protocol_output` and `protocol_input`.

Bugfixes

- PR #94: Fixes exception when testing equality of `ProtocolPath` objects.
- PR #100: Fixes precision issues when ensuring mole fractions are <= *1.0*.
- PR #102: Fixes replicated input for children of replicated protocols.
- PR #105: Fixes excess properties weighting by the wrong mole fractions.
- PR #107: Fixes excess properties being converged to the wrong uncertainty.
- PR #108: Fixes calculating MBAR gradients of reweighted properties.

Breaking Changes

- PR #98: `Substance.get_amount` renamed to `Substance.get_amounts` and now returns an immutable `frozenset` of `Amount` objects, rather than a single `Amount`.
- PR #104: The `DivideGradientByScalar`, `MultiplyGradientByScalar`, `AddGradients`, `SubtractGradients` and `WeightGradientByMoleFraction` protocols have been removed. The `WeightQuantityByMoleFraction` protocol has been renamed to `WeightByMoleFraction`.

Migration Guide

This release contained several public API breaking changes. For the most part, these can be remedied by the follow steps:

- Change all instances of `Substance.get_amount` to `Substance.get_amounts` and handle the newly returned frozenset of amounts, rather than the previously returned single amount.
- Replace the now removed protocols as follows:
  - `DivideGradientByScalar` -> `DivideValue`
  - `MultiplyGradientByScalar` -> `MultiplyValue`
  - `AddGradients` -> `AddValues`
  - `SubtractGradients` -> `SubtractValues`
  - `WeightGradientByMoleFraction` -> `WeightByMoleFraction`
  - `WeightQuantityByMoleFraction` -> `WeightByMoleFraction`

2.33.24 0.0.2 - Replicator Quick Fixes

A minor release to fix a number of minor bugs related to replicating protocols.

Bugfixes

- PR #90: Fixes merging gradient protocols with the same id.
- PR #92: Fixes replicating protocols for more than 10 template values.
- PR #93: Fixes `ConditionalGroup` objects losing their conditions input.

2.33.25 0.0.1 - Initial Release

The initial pre-alpha release of the framework.

2.34 Release Process

This document aims to outline the steps needed to release the `openff-evaluator` on `conda-forge`. This should only be done with the approval of the core maintainers.

2.34.1 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 - Descriptive Title
------------------------------

This release...

New Features
""""""""""""

* PR #X: Feature summary

Bugfixes
""""""""

* PR #Y: Fix Summary

Breaking Changes
""""""""""""""""

* PR #Z: Descriptive summary of the breaking change

Migration Guide
"""""""""""""""

This release contained several public API breaking changes. For the most part, these can␣
↪be
remedied by the follow steps:

* A somewhat verbose guide on how users should upgrade their code given the new breaking␣
↪changes.
```

2.34.2 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) Reformat the release notes from part 1) into markdown and paste into the description box.

a) Append the following extra message above the *New Features* title:

```
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/
↪install.html).

Please report bugs, request features, or ask questions through our
[issue tracker](https://github.com/openforcefield/openff-evaluator/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.**
```

Note - You do not need to upload any files. The source code will automatically be added as a `.tar.gz` file.

2.34.3 3: Trigger a New Build on Conda Forge

To trigger the build on `conda-forge`:

1) Create a fork of the openff-evaluator-feedstock and make the following changes to the `recipe/meta.yaml` file:

a) Update the `version` to match the release.

b) Set `build` to 0

c) Update any dependencies in the `requirements` section

d) Update the sha256 hash to the output of `curl -sL https://github.com/openforcefield/openff-evaluator/archive/{{ version }}.tar.gz | openssl sha256`

2) Open PR to merge the fork into the main feedstock:

a) The PR title should have the format `Release X.Y.Z`

b) No PR body text is needed

c) The CI will run on this PR (~30 minutes) and attempt to build the package.

d) If the build is successful the PR should be reviewed and merged by the feedstock maintainers.

e) **Once merged** the package is built again on and uploaded to anaconda.

3) Test the `conda-forge` package:

a) `conda install -c conda-forge openff-evaluator`

2.34.4 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**.

BIBLIOGRAPHY

[1] Alice Glättli, Xavier Daura, and Wilfred F van Gunsteren. Derivation of an improved simple point charge model for liquid water: spc/a and spc/l. *The Journal of chemical physics*, 116(22):9811–9828, 2002.

[2] Kyle A Beauchamp, Julie M Behr, Ariën S Rustenburg, Christopher I Bayly, Kenneth Kroenlein, and John D Chodera. Toward automated benchmarking of atomistic force fields: neat liquid densities and static dielectric constants from the thermoml data archive. *The Journal of Physical Chemistry B*, 119(40):12912–12920, 2015.

[3] Junmei Wang and Tingjun Hou. Application of molecular dynamics simulations in molecular property prediction. 1. density and heat of vaporization. *Journal of chemical theory and computation*, 7(7):2151–2165, 2011.

[4] David R Slochower, Niel M Henriksen, Lee-Ping Wang, John D Chodera, David L Mobley, and Michael K Gilson. Binding thermodynamics of host–guest systems with smirnoff99frosst 1.0. 5 from the open force field initiative. *Journal of Chemical Theory and Computation*, 15(11):6225–6242, 2019.

[1] John D Chodera. A simple method for automated equilibration detection in molecular simulations. *Journal of chemical theory and computation*, 12(4):1799–1805, 2016.

[2] Richard A Messerly, S Mostafa Razavi, and Michael R Shirts. Configuration-sampling-based surrogate models for rapid parameterization of non-bonded interactions. *Journal of Chemical Theory and Computation*, 14(6):3144–3162, 2018.

[1] Lee-Ping Wang, Teresa Head-Gordon, Jay W Ponder, Pengyu Ren, John D Chodera, Peter K Eastman, Todd J Martinez, and Vijay S Pande. Systematic improvement of a classical molecular model of water. *The Journal of Physical Chemistry B*, 117(34):9956–9972, 2013.

INDEX