OpenFF Toolkit Documentation

Release 0.10.0+75.g7b1a97f3.dirty

Open Force Field Consortium

Oct 13, 2021

GETTING STARTED

A modern, extensible library for molecular mechanics force field science from the Open Force Field Initiative

CHAPTER

ONE

INSTALLATION

1.1 Installing via conda

The simplest way to install the Open Force Field Toolkit is via the conda package manager. We publish packages via `conda-forge`.

If you are using the Anaconda scientific Python distribution, you already have the `conda` package manager installed. If not, the quickest way to get started is to install the Miniconda distribution, a lightweight, minimal installation of Python and the Conda package manager. See the conda documentation for detailed installation instructions. We recommend Miniforge, a drop-in replacement for Miniconda that uses the community-run `conda-forge` channel by default.

Once Conda is installed, use it to install the OpenFF Toolkit:

```
$ conda install -c conda-forge openff-toolkit
```

Note: Installation via the Conda package manager is the preferred method since all dependencies are automatically fetched and installed for you.

1.1.1 OS support

The OpenFF Toolkit is pure Python, and we expect it to work on any platform that supports its dependencies. Our automated testing takes place on both MacOS and Ubuntu Linux. For Windows support, we recommend using the Windows Subsystem for Linux (WSL) to run a Linux system integrated into Windows. We strongly suggest using WSL2, if your hardware supports it, for a smoother experience. WSL2 requires virtualization support in hardware. This is available on most modern CPUs, but may require activation in the BIOS.

Once WSL is configured, installing and using the Toolkit is done exactly as it would be for Linux. Note that by default, Jupyter Notebook will not be able to open a browser window and will log an error on startup; just ignore the error and open the link it provides in your ordinary Windows web browser.

Note: WSL2 does support GPU compute, at least with nvidia cards, but setting it up takes some work.

1.1.2 Conda environments

Conda environments that mix packages from the default channels and `conda-forge` can become inconsistent; to prevent this mixing, we recommend using `conda-forge` for all packages. The easiest way to do this is to install Conda with Miniforge.

If you already have a complex environment, or you wish to install a version of the Toolkit that is incompatible with other software you have installed, you can install the Toolkit into a new environment:

```
$ conda create -c conda-forge --name offtk openff-toolkit
```

An environment must be activated in any new shell session to use the software installed in it:

```
$ conda activate offtk
```

1.1.3 Upgrading your installation

To update an earlier `conda` installation of `openff-toolkit` to the latest release version, you can use `conda update`:

```
$ conda update -c conda-forge openff-toolkit
```

Note that this may update other packages or install new packages if the most recent release of the Toolkit requires it.

1.2 Installing from source

The OpenFF Toolkit has a lot of dependencies, so we strongly encourage installation with a package manager. The *developer's guide* describes setting up a development environment. If you're sure you want to install from source, check the `conda-forge` recipe for current dependencies, install them, download and extract the source distribution from GitHub, and then run `setup.py`:

```
$ cd openff-toolkit
$ python setup.py install
```

1.3 Single-file installer

As of release 0.4.1, single-file installers are available for each Open Force Field Toolkit release. These are provided primarily for users who do not have access to the Anaconda cloud for installing packages. These installers have few requirements beyond a Linux or MacOS operating system and will, in one command, produce a functional Python executable containing the Open Force Field Toolkit, as well as all required dependencies. The installers are very similar to the widely-used Miniconda `*.sh` files. Accordingly, installation using the "single-file installer" does not require root access.

The installers are between 200 and 300 MB each, and can be downloaded from the "Assets" section of the Toolkit's GitHub Releases page. They are generated using a workflow leveraging the Conda Constructor utility.

Please report any installer difficulties to the OFF Toolkit issue tracker, as we hope to make this a major distribution channel for the toolkit moving forward.

1.3.1 Installation

Download the appropriate installer (`openff-toolkit-<X.Y.Z>-<py3x>-<your platform>-x86_64.sh.zip`) from the "Assets" section at the bottom of the desired release on GitHub. Then, install the toolkit with the following command:

```
$ bash openff-toolkit-<X.Y.Z>-py37-<your platform>-x86_64.sh
```

and follow the prompts.

Note: You must have write access to the installation directory. This is generally somewhere in the user's home directory. When prompted, we recommend NOT making modifications to your `bash_profile`.

Warning: We recommend that you do not install this package as root. Conda is intended to support on-the-fly creation of several independent environments, and managing a multi-user Conda installation is complicated.

1.3.2 Usage

Any time you want to use this Conda environment in a terminal, run

```
$ source <install_directory>/etc/profile.d/conda.sh
$ conda activate base
```

Once the `base` environment is activated, your system will default to use Python (and other executables) from the newly installed Conda environment. For more information about Conda environments, see *Conda environments*

1.4 Optional dependencies (toolkits)

The OpenFF Toolkit outsources many common computational chemistry algorithms to other toolkits. Only one such toolkit is needed to gain access to all of the OpenFF Toolkit's features. If more than one is available, the Toolkit allows the user to specify their preference with the `toolkit_registry` argument to most functions and methods.

The `openff-toolkit` package installs everything needed to run the toolkit, including the optional dependencies RDKit and AmberTools. To install only the hard dependencies and provide your own optional dependencies, install the `openff-toolkit-base` package.

The OpenFF Toolkit requires an external toolkit for most functions. Though a `builtin` toolkit is provided, it implements only a small number of functions and is intended primarily for testing.

There are certain differences in toolkit behavior between RDKit/AmberTools and OpenEye when reading a small fraction of molecules, and we encourage you to report any unexpected behavior that may be caused by toolkit differences to our issue tracker.

1.4.1 RDKit

RDKit is a free and open source chemistry toolkit installed by default with the `openff-toolkit` package. It provides most of the functionality that the OpenFF Toolkit relies on.

1.4.2 AmberTools

AmberTools is a collection of free tools provided with the Amber MD software and installed by default with the `openff-toolkit` package. It provides a free implementation of functionality required by OpenFF Toolkit and not provided by RDKit.

1.4.3 OpenEye

The OpenFF Toolkit can optionally make use of the OpenEye toolkit if the user has a license key installed. Academic laboratories intending to release results into the public domain can obtain a free license key, while other users (including academics intending to use the software for purposes of generating protected intellectual property) must pay to obtain a license.

To install the OpenEye toolkits:

```
$ conda install -c openeye -c conda-forge openeye-toolkits
```

Though OpenEye can be installed for free, using it requires a license file. No essential `openff-toolkit` release capabilities *require* the OpenEye toolkit, but the Open Force Field developers make use of it in parameterizing new open source force fields.

CHAPTER

TWO

EXAMPLES USING SMIRNOFF WITH THE TOOLKIT

The following examples are available in the OpenFF toolkit repository. Each can be run interactively in the browser with binder, without installing anything on your computer.

2.1 Index of provided examples

- toolkit_showcase - parameterize a protein-ligand system with an OpenFF force field, simulate the resulting system, and visualize the result in the notebook
- forcefield_modification - modify force field parameters and evaluate how system energy changes
- conformer_energies - compute conformer energies of one or more small molecules using a SMIRNOFF force field
- SMIRNOFF_simulation - simulation of a molecule in the gas phase with the SMIRNOFF force field format
- forcefield_modification - modify force field parameters and evaluate how system energy changes
- using_smirnoff_in_amber_or_gromacs - convert a System generated with the Open Force Field Toolkit, which can be simulated natively with OpenMM, into AMBER prmtop/inpcrd and GROMACS top/gro input files through the ParmEd library.
- swap_amber_parameters - take a prepared AMBER protein-ligand system (prmtop and crd) along with a structure file of the ligand, and replace ligand parameters with OpenFF parameters.
- inspect_assigned_parameters - check which parameters are used in which molecules and generate parameter usage statistics.
- using_smirnoff_with_amber_protein_forcefield - use SMIRNOFF parameters for small molecules in combination with more conventional force fields for proteins and other components of your system (using ParmEd to combine parameterized structures)
- check_dataset_parameter_coverage - shows how to use the Open Force Field Toolkit to ingest a dataset of molecules, and generate a report summarizing any chemistry that can not be parameterized.
- visualization - shows how rich representation of `Molecule` objects work in the context of Jupyter Notebooks.

CHAPTER

THREE

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

3.1 Current development

- PR #964: Adds initial implementation of atom metadata dictionaries.
- PR #1097: Deprecates TopologyMolecule.
- PR #1097: Topology.from_openmmis no longer guaranteed to maintain the ordering of bonds, but now explicitly guarantees that it maintains the order of atoms (Neither of these ordering guarantees were explicitly documented before, but this may be a change from the previous behavior).

3.1.1 New features

- PR #1050: Conformer generation failures in `OpenEyeToolkitWrapper.generate_conformers`, and `RDKitToolkitWrapper.generate_conformers` now each raise `openff.toolkit.utils.exceptions.ConformerGenerationError` if conformer generation fails. The same behavior occurs in `Molecule.generate_conformers`, but only when the `toolkit_registry` argument is a `ToolkitWrapper`, not when it is a `ToolkitRegistry`. See also an entry in the "Behavior changed" section.
- PR #1036: SMARTS matching logic for library charges was updated to use only one unique match instead of enumerating all possible matches. This results in faster matching, particularly with larger moldules.
- PR #1001: Revamped the `Molecule.visualize()` method's `rdkit` backend for more pleasing and idiomatic 2D visualization by default.

3.1.2 Bugfixes

- PR #1052: Fixes Issue #986 by raising a subclass of `AttributeError` in `_ParameterAttributeHandler.__getattr__`
- PR #1030: Fixes a bug in which capitalization of the `bond_order_model` sometimes matters.

3.1.3 Behavior changed

- PR #1050: In `Molecule.generate_conformers`, a single toolkit wrapper failing to generate conformers is no longer fatal, but if all wrappers in a registry fail, then a `ValueError` will be raised. This mirrors the behavior of `Molecule.assign_partial_charges`.
- PR #1046: Changes OFFXML output to replace tabs with 4 spaces to standardize representation in different text viewers.
- PR #1036: SMARTS matching logic for library charges was updated to use only one unique match. No adverse side effects were found in testing, but could bad behavior may possibly exist in some unknown caes. Note that the default behavior for other parameter handlers was not updated.
- PR #1001: RDKit `Mol` objects created through the `Molecule.to_rdkit()` method have the `NoImplicit` property set to `True` on all atoms. This prevents RDKit from incorrectly adding hydrogen atoms to to molecule.
- PR #1058: Removes the unimplemented methods `ForceField.create_parmed_structure`, `Topology.to_parmed`, and `Topology.from_parmed`.
- PR #1065: The example `conformer_energies.py` script now uses the Sage 2.0.0 force field.

3.1.4 Tests updated

- PR #1017: Ensures that OpenEye-only CI builds really do lack both AmberTools and RDKit.

3.1.5 Improved documentation and warnings

- PR #1065: Example notebooks were updated to use the Sage Open Force Field
- PR #1062: Rewrote installation guide for clarity and comprehensiveness.

3.2 0.10.0 Improvements for force field fitting

3.2.1 Behaviors changed

- PR #1021: Renames `openff.toolkit.utils.exceptions.ParseError` to `openff.toolkit.utils.exceptions.SMILESParseError` to avoid a conflict with an identically-named exception in the SMIRNOFF XML parsing code.
- PR #1021: Renames and moves `openff.toolkit.typing.engines.smirnoff.forcefield.ParseError` to `openff.toolkit.utils.exceptions.SMIRNOFFParseError`. This `ParseError` is deprecated and will be removed in a future release.

3.2.2 New features and behaviors changed

- PR #1027: Corrects interconversion of Molecule objects with OEMol objects by ensuring atom names are correctly accessible via the `OEAtomBase.GetName()` and `OEAtomBase.SetName()` methods, rather that the non-standard `OEAtomBase.GetData("name")` and `OEAtomBase.SetData("name", name)`.
- PR #1007: Resolves Issue #456 by adding the `normalize_partial_charges` (default is `True`) keyword argument to `Molecule.assign_partial_charges`, `AmberToolsToolkitWrapper.assign_partial_charges`,`OpenEyeToolkitWrapper.assign_partial_charges`, `RDKitToolkitWrapper.assign_partial_charges`, and `BuiltInToolkitWrapper.assign_partial_charges`. This adds an offset to each atom's partial charge to ensure that their sum is equal to the net charge on the molecule (to the limit of a python float's precision, generally less than 1e-6 electron charge). **Note that, because this new behavior is ON by default, it may slightly affect the partial charges and energies of systems generated by running `create_openmm_system`.**
- PR #954: Adds `LibraryChargeType.from_molecule` which returns a `LibraryChargeType` object that will match the full molecule being parameterized, and assign it the same partial charges as are set on the input molecule.
- PR #923: Adds `Molecule.nth_degree_neighbors`, `Topology.nth_degree_neighbors`, `TopologyMolecule.nth_degree_neighbors`, which returns pairs of atoms that are separated in a molecule or topology by *exactly* N atoms.
- PR #917: `ForceField.create_openmm_system` now ensures that the cutoff of the `NonbondedForce` is set to the cutoff of the `vdWHandler` when it and a `Electrostatics` handler are present in the force field.
- PR #850: `OpenEyeToolkitWrapper.is_available` now returns `True` if *any* OpenEye tools are licensed (and installed). This allows, i.e, use of functionality that requires `OEChem` without having an `OEOmega` license.
- PR #909: Virtual site positions can now be computed directly in the toolkit. This functionality is accessed through
  - `FrozenMolecule.compute_virtual_site_positions_from_conformer`
  - `VirtualSite.compute_positions_from_conformer`
  - `VirtualParticle.compute_position_from_conformer`
  - `FrozenMolecule.compute_virtual_site_positions_from_atom_positions`
  - `VirtualSite.compute_positions_from_atom_positions`
  - `VirtualParticle.compute_position_from_atom_positions` where the positions can be computed from a stored conformer, or an input vector of atom positions.
  - Tests have been added (`TestMolecule.test_*_virtual_site_position`) to check for sane behavior. The tests do not directly compare OpenMM position equivalence, but offline tests show that they are equivalent.
  - The helper method `VirtualSiteHandler.create_openff_virtual_sites` is now public, which returns a modified topology with virtual sites added.
  - Virtual sites now expose the parameters used to create its local frame via the read-only properties
    - `VirtualSite.local_frame_weights`
    - `VirtualSite.local_frame_position`
  - Adding virtual sites via the `Molecule` API now have defaults for `sigma`, `epsilon`, and `charge_increment` set to 0 with appropriate units, rather than `None`

- PR #956: Added `ForceField.get_partial_charges()` to more easily compute the partial charges assigned by a force field for a molecule.
- PR #1006: Two behavior changes in the SMILES output for `to_file()` and `to_file_obj()`:
  - The RDKit and OpenEye wrappers now output the same SMILES as `to_smiles()`. This uses explicit hydrogens rather than the toolkit's default of implicit hydrogens.
  - The RDKit wrapper no longer includes a header line. This improves the consistency between the OpenEye and RDKit outputs.

3.2.3 Bugfixes

- PR #1024: Small changes for compatibility with OpenMM 7.6.
- PR #1003: Fixes Issue #1000, where a stereochemistry warning is sometimes erroneously emitted when loading a stereogenic molecule using `Molecule.from_pdb_and_smiles`
- PR #1002: Fixes a bug in which OFFXML files could inadvertently be loaded from subdirectories.
- PR #969: Fixes a bug in which the cutoff distance of the `NonbondedForce` generated by `ForceField.create_openmm_system` was not set to the value specified by the vdW and Electrostatics handlers.
- PR #909: Fixed several bugs related to creating an OpenMM system with virtual sites created via the `Molecule` virtual site API
- PR #1006: Many small fixes to the toolkit wrapper I/O for better error handling, improved consistency between reading from a file vs. file object, and improved consistency between the RDKit and OEChem toolkit wrappers. For the full list see Issue #1005. Some of the more significant fixes are:
  - `RDKitToolkitWrapper.from_file_obj()` now uses the same structure normaliation as `from_file()`.
  - `from_smiles()` now raises an `openff.toolkit.utils.exceptions.SMILESParsingError` if the SMILES could not be parsed.
  - OEChem input and output files now raise an OSError if the file could not be opened.
  - All input file object readers now support file objects open in binary mode.

3.2.4 Examples added

- PR #763: Adds an introductory example showcasing the toolkit parameterizing a protein-ligand simulation.
- PR #955: Refreshed the force field modification example
- PR #934 and conda-forge/openff-toolkit-feedstock#9: Added `openff-toolkit-examples` Conda package for easy installation of examples and their dependencies. Simply `conda install -c conda-forge openff-toolkit-examples` and then run the `openff-toolkit-examples` script to copy the examples suite to a convenient place to run them!

3.2.5 Tests updated

- PR #963: Several tests modules used functions from test_forcefield.py that created an OpenFF Molecule without a toolkit. These functions are now in their own module so they can be imported directly, without the overhead of going through test_forcefield.
- PR #997: Several XML snippets in `test_forcefield.py` that were scattered around inside of classes and functions are now moved to the module level.

3.3 0.9.2 Minor feature and bugfix release

3.3.1 New features and behaviors changed

- PR #762: `Molecule.from_rdkit` now converts implicit hydrogens into explicit hydrogens by default. This change may affect `RDKitToolkitWrapper/Molecule.from_smiles`, `from_mapped_smiles`, `from_file`, `from_file_obj`, `from_inchi`, and `from_qcschema`. This new behavior can be disabled using the `hydrogens_are_explicit=True` keyword argument to `from_smiles`, or loading the molecule into the desired explicit protonation state in RDKit, then calling `from_rdkit` on the RDKit molecule with `hydrogens_are_explicit=True`.
- PR #894: Calls to `Molecule.from_openeye`, `Molecule.from_rdkit`, `Molecule.from_smiles`, `OpenEyeToolkitWrapper.from_smiles`, and `RDKitToolkitWrapper.from_smiles` will now load atom maps into the the resulting `Molecule's offmol.properties['atom_map']` field, even if not all atoms have map indices assigned.
- PR #904: `TopologyAtom` objects now have an element getter `TopologyAtom.element`.

3.3.2 Bugfixes

- PR #891: Calls to `Molecule/OpenEyeToolkitWrapper.from_openeye` no longer mutate the input OE molecule.
- PR #897: Fixes enumeration of stereoisomers for molecules with already defined stereochemistry using `RDKitToolkitWrapper.enumerate_stereoisomers`.
- PR #859: Makes `RDKitToolkitWrapper.enumerate_tautomers` actually use the `max_states` keyword argument during tautomer generation, which will reduce resource use in some cases.

3.3.3 Improved documentation and warnings

- PR #862: Clarify that `System` objects produced by the toolkit are OpenMM `Systems` in anticipation of forthcoming OpenFF `Systems`. Fixes Issue #618.
- PR #863: Documented how to build the docs in the developers guide.
- PR #870: Reorganised documentation to improve discoverability and allow future additions.
- PR #871: Changed Markdown parser from m2r2 to MyST for improved documentation rendering.
- PR #880: Cleanup and partial rewrite of the developer's guide.
- PR #906: Cleaner instructions on how to setup development environment.

0.9.1 - Minor feature and bugfix release

New features

- PR #839: Add support for computing WBOs from multiple conformers using the AmberTools and OpenEye toolkits, and from ELF10 conformers using the OpenEye toolkit wrapper.
- PR #832: Expose ELF conformer selection through the `Molecule` API via a new `apply_elf_conformer_selection` function.
- PR #831: Expose ELF conformer selection through the OpenEye wrapper.
- PR #790: Fixes Issue #720 where qcschema roundtrip to/from results in an error due to missing cmiles entry in attributes.
- PR #793: Add an initial ELF conformer selection implementation which uses RDKit.
- PR #799: Closes Issue #746 by adding `Molecule.smirnoff_impropers`, `Molecule.amber_impropers`, `TopologyMolecule.smirnoff_impropers`, `TopologyMolecule.amber_impropers`, `Topology.smirnoff_impropers`, and `Topology.amber_impropers`.
- PR #847: Instances of `ParameterAttribute` documentation can now specify their docstrings with the optional `docstring` argument to the `__init__()` method.
- PR #827: The setter for `Topology.box_vectors` now infers box vectors when box lengths are pass as a list of length 3.

Behavior changed

- PR #802: Fixes Issue #408. The 1-4 scaling factor for electrostatic interactions is now properly set by the value specified in the force field. Previously it fell back to a default value of 0.83333. The toolkit may now produce slightly different energies as a result of this change.
- PR #839: The average WBO will now be returned when multiple conformers are provided to `assign_fractional_bond_orders` using `use_conformers`.
- PR #816: Force field file paths are now loaded in a case-insensitive manner.

Bugfixes

- PR #849: Changes `create_openmm_system` so that it no longer uses the conformers on existing reference molecules (if present) to calculate Wiberg bond orders. Instead, new conformers are always generated during parameterization.

Improved documentation and warnings

- PR #838: Corrects spacing of "forcefield" to "force field" throughout documentation. Fixes Issue #112.
- PR #846: Corrects dead links throughout release history. Fixes Issue #835.
- PR #847: Documentation now compiles with far fewer warnings, and in many cases more correctly. Additionally, `ParameterAttribute` documentation no longer appears incorrectly in classes where it is used. Fixes Issue #397.

0.9.0 - Namespace Migration

This release marks the transition from the old `openforcefield` branding over to its new identity as `openff-toolkit`. This change has been made to better represent the role of the toolkit, and highlight its place in the larger Open Force Field (OpenFF) ecosystem.

From version `0.9.0` onwards the toolkit will need to be imported as `import openff.toolkit.XXX` and `from openff.toolkit import XXX`.

API-breaking changes

- PR #803: Migrates `openforcefield` imports to `openff.toolkit`.

0.8.4 - Minor feature and bugfix release

This release is intended to be functionally identical to 0.9.1. The only difference is that it uses the "openforcefield" namespace.

This release is a final patch for the `0.8.X` series of releases of the toolkit, and also marks the last version of the toolkit which will be imported as `import openforcefield.XXX` / `from openforcefield import XXX`. From version `0.9.0` onwards the toolkit will be importable only as `import openff.toolkit.XXX` / `from openff.toolkit import XXX`.

Note This change will also be accompanied by a renaming of the package from `openforcefield` to `openff-toolkit`, so users need not worry about accidentally pulling in a version with changed imports. Users will have to explicitly choose to install the `openff-toolkit` package once released which will contain the breaking import changes.

0.8.3 - Major bugfix release

This release fixes a critical bug in van der Waals parameter assignment.

This release is also a final patch for the `0.8.X` series of releases of the toolkit, and also marks the last version of the toolkit which will be imported as `import openforcefield.XXX` / `from openforcefield import XXX`. From version `0.9.0` onwards the toolkit will be importable only as `import openff.toolkit.XXX` / `from openff.toolkit import XXX`.

Note This change will also be accompanied by a renaming of the package from `openforcefield` to `openff-toolkit`, so users need not worry about accidentally pulling in a version with changed imports. Users will have to explicitly choose to install the `openff-toolkit` package once released which will contain the breaking import changes.

Bugfixes

- PR #808: Fixes Issue #807, which tracks a major bug in the interconversion between a vdW `sigma` and `rmin_half` parameter.

New features

- PR #794: Adds a decorator `@requires_package` that denotes a function requires an optional dependency.
- PR #805: Adds a deprecation warning for the up-coming release of the `openff-toolkit` package and its import breaking changes.

0.8.2 - Bugfix release

WARNING: This release was later found to contain a major bug, Issue #807, and produces incorrect energies.

Bugfixes

- PR #786: Fixes Issue #785 where RDKitToolkitWrapper would sometimes expect stereochemistry to be defined for non-stereogenic bonds when loading from SDF.
- PR #786: Fixes an issue where using the `Molecule` copy constructor (`newmol = Molecule(oldmol)`) would result in the copy sharing the same `.properties` dict as the original (as in, changes to the `.properties` dict of the copy would be reflected in the original).
- PR #789: Fixes a regression noted in Issue #788 where creating `vdWHandler.vdWType` or setting `sigma` or `rmin_half` using Quantities represented as strings resulted in an error.

0.8.1 - Bugfix and minor feature release

WARNING: This release was later found to contain a major bug, Issue #807, and produces incorrect energies.

API-breaking changes

- PR #757: Renames `test_forcefields/smirnoff99Frosst.offxml` to `test_forcefields/test_forcefield.offxml` to avoid confusion with any of the ACTUAL released FFs in the smirnoff99Frosst line
- PR #751: Removes the optional `oetools=("oechem", "oequacpac", "oeiupac", "oeomega")` keyword argument from `OpenEyeToolkitWrapper.is_available`, as there are no special behaviors that are accessed in the case of partially-licensed OpenEye backends. The new behavior of this method is the same as if the default value above is always provided.

Behavior Changed

- PR #583: Methods such as `Molecule.from_rdkit` and `Molecule.from_openeye`, which delegate their internal logic to `ToolkitRegistry` functions, now guarantee that they will return an object of the correct type when being called on `Molecule`-derived classes. Previously, running these constructors using subclasses of `FrozenMolecule` would not return an instance of that subclass, but rather just an instance of a `Molecule`.
- PR #753: `ParameterLookupError` is now raised when passing to `ParameterList.index` a SMIRKS pattern not found in the parameter list.

New features

- PR #751: Adds `LicenseError`, a subclass of `ToolkitUnavailableException` which is raised when attempting to add a cheminformatics `ToolkitWrapper` for a toolkit that is installed but unlicensed.
- PR #678: Adds `ForceField.deregister_parameter_handler`.
- PR #730: Adds `Topology.is_periodic`.
- PR #753: Adds `ParameterHandler.__getitem__` to look up individual `ParameterType` objects.

Bugfixes

- PR #745: Fixes bug when serializing molecule with conformers to JSON.
- PR #750: Fixes a bug causing either `sigma` or `rmin_half` to sometimes be missing on `vdWHandler.vdWType` objects.
- PR #756: Fixes bug when running `vdWHandler.create_force` using a `vdWHandler` that was initialized using the API.
- PR #776: Fixes a bug in which the `Topology.from_openmm` and `Topology.from_mdtraj` methods would dangerously allow `unique_molecules=None`.
- PR #777: `RDKitToolkitWrapper` now outputs the full warning message when `allow_undefined_stereo=True` (previously the description of which stereo was undefined was squelched)

0.8.0 - Virtual Sites

Major Feature: Support for the SMIRNOFF VirtualSite tag

This release implements the SMIRNOFF virtual site specification. The implementation enables support for models using off-site charges, including 4- and 5-point water models, in addition to lone pair modeling on various functional groups. The primary focus was on the ability to parameterize a system using virtual sites, and generating an OpenMM system with all virtual sites present and ready for evaluation. Support for formats other than OpenMM has not be implemented in this release, but may come with the appearance of the OpenFF system object. In addition to implementing the specification, the toolkit `Molecule` objects now allow the creation and manipulation of virtual sites.

This change is documented in the Virtual sites page of the user guide.

Minor Feature: Support for the 0.4 ChargeIncrementModel tag

To allow for more convenient fitting of `ChargeIncrement` parameters, it is now possible to specify one less `charge_increment` value than there are tagged atoms in a `ChargeIncrement`'s `smirks`. The missing `charge_increment` value will be calculated at parameterization-time to make the sum of the charge contributions from a `ChargeIncrement` parameter equal to zero. Since this change allows for force fields that are incompatible with the previous specification, this new style of `ChargeIncrement` must specify a `ChargeIncrementModel` section version of `0.4`. All `0.3`-compatible `ChargeIncrement` parameters are compatible with the `0.4` `ChargeIncrementModel` specification.

More details and examples of this change are available in The ChargeIncrementModel tag in the SMIRNOFF specification

New features

- PR #726: Adds support for the 0.4 ChargeIncrementModel spec, allowing for the specification of one fewer `charge_increment` values than there are tagged atoms in the `smirks`, and automatically assigning the final atom an offsetting charge.
- PR #548: Adds support for the `VirtualSites` tag in the SMIRNOFF specification
- PR #548: Adds `replace` and `all_permutations` kwarg to
  - `Molecule.add_bond_charge_virtual_site`
  - `Molecule.add_monovalent_lone_pair_virtual_site`
  - `Molecule.add_divalent_lone_pair_virtual_site`
  - `Molecule.add_trivalent_lone_pair_virtual_site`
- PR #548: Adds `orientations` to
  - `BondChargeVirtualSite`
  - `MonovalentLonePairVirtualSite`
  - `DivalentLonePairVirtualSite`
  - `TrivalentLonePairVirtualSite`
- PR #548: Adds
  - `VirtualParticle`
  - `TopologyVirtualParticle`
  - `BondChargeVirtualSite.get_openmm_virtual_site`
  - `MonovalentLonePairVirtualSite.get_openmm_virtual_site`
  - `DivalentLonePairVirtualSite.get_openmm_virtual_site`
  - `TrivalentLonePairVirtualSite.get_openmm_virtual_site`
  - `ValenceDict.key_transform`
  - `ValenceDict.index_of`
  - `ImproperDict.key_transform`
  - `ImproperDict.index_of`
- PR #705: Adds interpolation based on fractional bond orders for harmonic bonds. This includes interpolation for both the force constant `k` and/or equilibrium bond distance `length`. This is accompanied by a bump in the `<Bonds>` section of the SMIRNOFF spec (but not the entire spec).
- PR #718: Adds `.rings` and `.n_rings` to `Molecule` and `.is_in_ring` to `Atom` and `Bond`

Bugfixes

- PR #682: Catches failures in `Molecule.from_iupac` instead of silently failing.
- PR #743: Prevents the non-bonded (vdW) cutoff from silently falling back to the OpenMM default of 1 nm in `Forcefield.create_openmm_system` and instead sets its to the value specified by the force field.
- PR #737: Prevents OpenEye from incidentally being used in the conformer generation step of `AmberToolsToolkitWrapper.assign_fractional_bond_orders`.

Behavior changed

- PR #705: Changes the default values in the `<Bonds>` section of the SMIRNOFF spec to `fractional_bondorder_method="AM1-Wiberg"` and `potential="(k/2)*(r-length)^2"`, which is backwards-compatible with and equivalent to `potential="harmonic"`.

Examples added

- PR #548: Adds a virtual site example notebook to run an OpenMM simulation with virtual sites, and compares positions and potential energy of TIP5P water between OpenFF and OpenMM force fields.

API-breaking changes

- PR #548: Methods
  - `Molecule.add_bond_charge_virtual_site`
  - `Molecule.add_monovalent_lone_pair_virtual_site`
  - `Molecule.add_divalent_lone_pair_virtual_site`
  - `Molecule.add_trivalent_lone_pair_virtual_site` now only accept a list of atoms, not a list of integers, to define to parent atoms
- PR #548: Removes `VirtualParticle.molecule_particle_index`
- PR #548: Removes `outOfPlaneAngle` from
  - `DivalentLonePairVirtualSite`
  - `TrivalentLonePairVirtualSite`
- PR #548: Removes `inPlaneAngle` from `TrivalentLonePairVirtualSite`
- PR #548: Removes `weights` from
  - `BondChargeVirtualSite`
  - `MonovalentLonePairVirtualSite`
  - `DivalentLonePairVirtualSite`
  - `TrivalentLonePairVirtualSite`

Tests added

- PR #548: Adds test for
  - The virtual site parameter handler
  - TIP5P water dimer energy and positions
  - Adds tests to for virtual site/particle indexing/counting

0.7.2 - Bugfix and minor feature release

New features

- PR #662: Adds `.aromaticity_model` of `ForceField` and `.TAGNAME` of `ParameterHandler` as public attributes.
- PR #667 and PR #681 linted the codebase with `black` and `isort`, respectively.
- PR #675 adds `.toolkit_version` to `ToolkitWrapper` and `.registered_toolkit_versions` to `ToolkitRegistry`.
- PR #696 Exposes a setter for `ForceField.aromaticity_model`
- PR #685 Adds a custom `__hash__` function to `ForceField`

Behavior changed

- PR #684: Changes `ToolkitRegistry` to return an empty registry when initialized with no arguments, i.e. `ToolkitRegistry()` and makes the `register_imported_toolkit_wrappers` argument private.
- PR #711: The setter for `Topology.box_vectors` now infers box vectors (a 3x3 matrix) when box lengths (a 3x1 array) are passed, assuming an orthogonal box.
- PR #649: Makes SMARTS searches stereochemistry-specific (if stereo is specified in the SMARTS) for both OpenEye and RDKit backends. Also ensures molecule aromaticity is re-perceived according to the ForceField's specified aromaticity model, which may overwrite user-specified aromaticity on the `Molecule`
- PR #648: Removes the `utils.structure` module, which was deprecated in 0.2.0.
- PR #670: Makes the `Topology` returned by `create_openmm_system` contain the partial charges and partial bond orders (if any) assigned during parameterization.
- PR #675 changes the exception raised when no `antechamber` executable is found from `IOError` to `AntechamberNotFoundError`
- PR #696 Adds an `aromaticity_model` keyword argument to the `ForceField` constructor, which defaults to `DEFAULT_AROMATICITY_MODEL`.

Bugfixes

- PR #715: Closes issue Issue #475 writing a "PDB" file using OE backend rearranges the order of the atoms by pushing the hydrogens to the bottom.
- PR #649: Prevents 2020 OE toolkit from issuing a warning caused by doing stereo-specific smarts searches on certain structures.
- PR #724: Closes issue Issue #502 Adding a utility function Topology.to_file() to write topology and positions to a "PDB" file using openmm backend for pdb file write.

Tests added

- PR #694: Adds automated testing to code snippets in docs.
- PR #715: Adds tests for pdb file writes using OE backend.
- PR #724: Adds tests for the utility function Topology.to_file().

0.7.1 - OETK2020 Compatibility and Minor Update

This is the first of our patch releases on our new planned monthly release schedule.

Detailed release notes are below, but the major new features of this release are updates for compatibility with the new 2020 OpenEye Toolkits release, the `get_available_force_fields` function, and the disregarding of pyrimidal nitrogen stereochemistry in molecule isomorphism checks.

Behavior changed

- PR #646: Checking for `Molecule` equality using the `==` operator now disregards all pyrimidal nitrogen stereochemistry by default. To re-enable, use `Molecule.{is|are}_isomorphic` with the `strip_pyrimidal_n_atom_stereo=False` keyword argument.
- PR #646: Adds an optional `toolkit_registry` keyword argument to `Molecule.are_isomorphic`, which identifies the toolkit that should be used to search for pyrimidal nitrogens.

Bugfixes

- PR #647: Updates `OpenEyeToolkitWrapper` for 2020.0.4 OpenEye Toolkit behavior/API changes.
- PR #646: Fixes a bug where `Molecule.chemical_environment_matches` was not able to accept a `ChemicalEnvironment` object as a query.
- PR #634: Fixes a bug in which calling `RDKitToolkitWrapper.from_file` directly would not load files correctly if passed lowercase `file_format`. Note that this bug did not occur when calling `Molecule.from_file`.
- PR #631: Fixes a bug in which calling `unit_to_string` returned `None` when the unit is dimensionless. Now `"dimensionless"` is returned.
- PR #630: Closes issue Issue #629 in which the wrong exception is raised when attempting to instantiate a `ForceField` from an unparsable string.

New features

- PR #632: Adds `ForceField.registered_parameter_handlers`
- PR #614: Adds `ToolkitRegistry.deregister_toolkit` to de-register registered toolkits, which can include toolkit wrappers loaded into `GLOBAL_TOOLKIT_REGISTRY` by default.
- PR #656: Adds a new allowed `am1elf10` option to the OpenEye implementation of `assign_partial_charges` which calculates the average partial charges at the AM1 level of theory using conformers selected using the ELF10 method.
- PR #643: Adds `openforcefield.typing.engines.smirnoff.forcefield.get_available_force_fields`, which returns paths to the files of force fields available through entry point plugins.

0.7.0 - Charge Increment Model, Proper Torsion interpolation, and new Molecule methods

This is a relatively large release, motivated by the idea that changing existing functionality is bad so we shouldn't do it too often, but when we do change things we should do it all at once.

Here's a brief rundown of what changed, migration tips, and how to find more details in the full release notes below:

- To provide more consistent partial charges for a given molecule, existing conformers are now disregarded by default by `Molecule.assign_partial_charges`. Instead, new conformers are generated for use in semiempirical calculations. Search for `use_conformers`.
- Formal charges are now always returned as `simtk.unit.Quantity` objects, with units of elementary charge. To convert them to integers, use `from simtk import unit` and `atom.formal_charge.value_in_unit(unit.elementary_charge)` or `mol.total_charge.value_in_unit(unit.elementary_charge)`. Search `atom.formal_charge`.
- The OpenFF Toolkit now automatically reads and writes partial charges in SDF files. Search for `atom.dprop.PartialCharges`.
- The OpenFF Toolkit now has different behavior for handling multi-molecule and multi-conformer SDF files. Search `multi-conformer`.
- The OpenFF Toolkit now distinguishes between partial charges that are all-zero and partial charges that are unknown. Search `partial_charges = None`.
- `Topology.to_openmm` now assigns unique atoms names by default. Search `ensure_unique_atom_names`.
- Molecule equality checks are now done by graph comparison instead of SMILES comparison. Search `Molecule.are_isomorphic`.
- The `ChemicalEnvironment` module was almost entirely removed, as it is an outdated duplicate of some Chemper functionality. Search `ChemicalEnvironment`.
- `TopologyMolecule.topology_particle_start_index` has been removed from the `TopologyMolecule` API, since atoms and virtualsites are no longer contiguous in the `Topology` particle indexing system. Search `topology_particle_start_index`.
- `compute_wiberg_bond_orders` has been renamed to `assign_fractional_bond_orders`.

There are also a number of new features, such as:

- Support for `ChargeIncrementModel` sections in force fields.
- Support for `ProperTorsion` `k` interpolation in force fields using fractional bond orders.

- Support for AM1-Mulliken, Gasteiger, and other charge methods using the new `assign_partial_charges` methods.
- Support for AM1-Wiberg bond order calculation using either the OpenEye or RDKit/AmberTools backends and the `assign_fractional_bond_orders` methods.
- Initial (limited) interoperability with QCArchive, via `Molecule.to_qcschema` and `from_qcschema`.
- A `Molecule.visualize` method.
- Several additional `Molecule` methods, including state enumeration and mapped SMILES creation.

Major Feature: Support for the SMIRNOFF ChargeIncrementModel tag

The ChargeIncrementModel tag in the SMIRNOFF specification provides analagous functionality to AM1-BCC, except that instead of AM1-Mulliken charges, a number of different charge methods can be called, and instead of a fixed library of two-atom charge corrections, an arbitrary number of SMIRKS-based, N-atom charge corrections can be defined in the SMIRNOFF format.

The initial implementation of the SMIRNOFF `ChargeIncrementModel` tag accepts keywords for `version`, `partial_charge_method`, and `number_of_conformers`. `partial_charge_method` can be any string, and it is up to the `ToolkitWrapper`'s `compute_partial_charges` methods to understand what they mean. For geometry-independent `partial_charge_method` choices, `number_of_conformers` should be set to zero.

SMIRKS-based parameter application for `ChargeIncrement` parameters is different than other SMIRNOFF sections. The initial implementation of `ChargeIncrementModelHandler` follows these rules:

- an atom can be subject to many `ChargeIncrement` parameters, which combine additively.
- a `ChargeIncrement` that matches a set of atoms is overwritten only if another `ChargeIncrement` matches the same group of atoms, regardless of order. This overriding follows the normal SMIRNOFF hierarchy.

To give a concise example, what if a molecule `A-B(-C)-D` were being parametrized, and the force field defined `ChargeIncrement` SMIRKS in the following order?

1) `[A:1]-[B:2]`
2) `[B:1]-[A:2]`
3) `[A:1]-[B:2]-[C:3]`
4) `[*:1]-[B:2](-[*:3])-[*:4]`
5) `[D:1]-[B:2](-[*:3])-[*:4]`

In the case above, the ChargeIncrement from parameters 1 and 4 would NOT be applied to the molecule, since another parameter matching the same set of atoms is specified further down in the parameter hierarchy (despite those subsequent matches being in a different order).

Ultimately, the ChargeIncrement contributions from parameters 2, 3, and 5 would be summed and applied.

It's also important to identify a behavior that these rules were written to *avoid*: if not for the "regardless of order" clause in the second rule, parameters 4 and 5 could actually have been applied six and two times, respectively (due to symmetry in the SMIRKS and the use of wildcards). This situation could also arise as a result of molecular symmetry. For example, a methyl group could match the SMIRKS `[C:1]([H:2])([H:3])([H:4])` six ways (with different orderings of the three hydrogen atoms), but the user would almost certainly not intend for the charge increments to be applied six times. The "regardless of order" clause was added specifically to address this.

In short, the first time a group of atoms becomes involved in a `ChargeIncrement` together, the OpenMM `System` gains a new parameter "slot". Only another `ChargeIncrement` which applies to the exact same group of atoms (in any order) can take over the "slot", pushing the original `ChargeIncrement` out.

Major Feature: Support for ProperTorsion k value interpolation

Chaya Stern's work showed that we may be able to produce higher-quality proper torsion parameters by taking into account the "partial bond order" of the torsion's central bond. We now have the machinery to compute AM1-Wiberg partial bond orders for entire molecules using the `assign_fractional_bond_orders` methods of either `OpenEyeToolkitWrapper` or `AmberToolsToolkitWrapper`. The thought is that, if some simple electron population analysis shows that a certain aromatic bond's order is 1.53, maybe rotations about that bond can be described well by interpolating 53% of the way between the single and double bond k values.

Full details of how to define a torsion-interpolating SMIRNOFF force fields are available in the ProperTorsions section of the SMIRNOFF specification.

Behavior changed

- PR #508: In order to provide the same results for the same chemical species, regardless of input conformation, `Molecule` `assign_partial_charges`, `compute_partial_charges_am1bcc`, and `assign_fractional_bond_orders` methods now default to ignore input conformers and generate new conformer(s) of the molecule before running semiempirical calculations. Users can override this behavior by specifying the keyword argument `use_conformers=molecule.conformers`.
- PR #281: Closes Issue #250 by adding support for partial charge I/O in SDF. The partial charges are stored as a property in the SDF molecule block under the tag `<atom.dprop.PartialCharge>`.
- PR #281: If a `Molecule`'s `partial_charges` attribute is set to `None` (the default value), calling `to_openeye` will now produce a OE molecule with partial charges set to `nan`. This would previously produce an OE molecule with partial charges of 0.0, which was a loss of information, since it wouldn't be clear whether the original OFFMol's partial charges were REALLY all-zero as opposed to `None`. OpenEye toolkit wrapper methods such as `from_smiles` and `from_file` now produce OFFMols with `partial_charges = None` when appropriate (previously these would produce OFFMols with all-zero charges, for the same reasoning as above).
- PR #281: `Molecule` `to_rdkit` now sets partial charges on the RDAtom's `PartialCharges` property (this was previously set on the `partial_charges` property). If the `Molecule`'s partial_charges attribute is `None`, this property will not be defined on the RDAtoms.
- PR #281: Enforce the behavior during SDF I/O that a SDF may contain multiple *molecules*, but that the OFF Toolkit does not assume that it contains multiple *conformers of the same molecule*. This is an important distinction, since otherwise there is ambiguity around whether properties of one entry in a SDF are shared among several molecule blocks or not, or how to resolve conflicts if properties are defined differently for several "conformers" of chemically-identical species (More info here). If the user requests the OFF Toolkit to write a multi-conformer `Molecule` to SDF, only the first conformer will be written. For more fine-grained control of writing properties, conformers, and partial charges, consider using `Molecule.to_rdkit` or `Molecule.to_openeye` and using the functionality offered by those packages.
- PR #281: Due to different constraints placed on the data types allowed by external toolkits, we make our best effort to preserve `Molecule` `properties` when converting molecules to other packages, but users should be aware that no guarantee of data integrity is made. The only data format for keys and values in the property dict that we will try to support through a roundtrip to another toolkit's Molecule object is `string`.
- PR #574: Removed check that all partial charges are zero after assignment by `quacpac` when AM1BCC used for charge assignment. This check fails erroneously for cases in which the partial charge assignments are correctly all zero, such as for `N#N`. It is also an unnecessary check given that `quacpac` will reliably indicate when it has failed to assign charges.
- PR #597: Energy-minimized sample systems with Parsley 1.1.0.

- PR #558: The `Topology` particle indexing system now orders `TopologyVirtualSites` after all atoms.
- PR #469: When running `Topology.to_openmm`, unique atom names are generated if the provided atom names are not unique (overriding any existing atom names). This uniqueness extends only to atoms in the same molecule. To disable this behavior, set the kwarg `ensure_unique_atom_names=False`.
- PR #472: `Molecule.__eq__` now uses the new `Molecule.are_isomorphic` to perform the similarity checking.
- PR #472: The `Topology.from_openmm` and `Topology.add_molecule` methods now use the `Molecule.are_isomorphic` method to match molecules.
- PR #551: Implemented the `ParameterHandler.get_parameter` function (would previously return `None`).

API-breaking changes

- PR #471: Closes Issue #465. `atom.formal_charge` and `molecule.total_charge` now return `simtk.unit.Quantity` objects instead of integers. To preserve backward compatibility, the setter for `atom.formal_charge` can accept either a `simtk.unit.Quantity` or an integer.
- PR #601: Removes almost all of the previous `ChemicalEnvironment` API, since this entire module was simply copied from Chemper several years ago and has fallen behind on updates. Currently only `ChemicalEnvironment.get_type`, `ChemicalEnvironment.validate`, and an equivalent classmethod `ChemicalEnvironment.validate_smirks` remain. Also, please comment on this GitHub issue if you HAVE been using the previous extra functionality in this module and would like us to prioritize creation of a Chemper conda package.
- PR #558: Removes `TopologyMolecule.topology_particle_start_index`, since the `Topology` particle indexing system now orders `TopologyVirtualSites` after all atoms. `TopologyMolecule.atom_start_topology_index` and `TopologyMolecule.virtual_particle_start_topology_index` are still available to access the appropriate values in the respective topology indexing systems.
- PR #508: `OpenEyeToolkitWrapper.compute_wiberg_bond_orders` is now `OpenEyeToolkitWrapper.assign_fractional_bond_orders`. The `charge_model` keyword is now `bond_order_model`. The allowed values of this keyword have changed from `am1` and `pm3` to `am1-wiberg` and `pm3-wiberg`, respectively.
- PR #508: `Molecule.compute_wiberg_bond_orders` is now `Molecule.assign_fractional_bond_orders`.
- PR #595: Removed functions `openforcefield.utils.utils.temporary_directory` and `openforcefield.utils.utils.temporary_cd` and replaced their behavior with `tempfile.TemporaryDirectory()`.

New features

- PR #471: Closes Issue #208 by implementing support for the `ChargeIncrementModel` tag in the SMIRNOFF specification.
- PR #471: Implements `Molecule.assign_partial_charges`, which calls one of the newly-implemented `OpenEyeToolkitWrapper.assign_partial_charges`, and `AmberToolsToolkitWrapper.assign_partial_charges`. `strict_n_conformers` is a optional boolean keyword argument indicating whether an `IncorrectNumConformersError` should be raised if an invalid number of conformers is supplied during partial charge calculation. For example, if two conformers are supplied, but `partial_charge_method="AM1BCC"` is also set, then there is no clear use for the second conformer. The previous behavior in this case was to raise a warning, and to preserve that behavior, `strict_n_conformers` defaults to a value of `False`.

- PR #471: Adds keyword argument `raise_exception_types` (default: `[Exception]`) to `ToolkitRegistry.call`. The default value will provide the previous OpenFF Toolkit behavior, which is that the first ToolkitWrapper that can provide the requested method is called, and it either returns on success or raises an exception. This new keyword argument allows the ToolkitRegistry to *ignore* certain exceptions, but treat others as fatal. If `raise_exception_types = []`, the ToolkitRegistry will attempt to call each ToolkitWrapper that provides the requested method and if none succeeds, a single `ValueError` will be raised, with text listing the errors that were raised by each ToolkitWrapper.
- PR #601: Adds `RDKitToolkitWrapper.get_tagged_smarts_connectivity` and `OpenEyeToolkitWrapper.get_tagged_smarts_connectivity`, which allow the use of either toolkit for smirks/tagged smarts validation.
- PR #600: Adds `ForceField.__getitem__` to look up `ParameterHandler` objects based on their string names.
- PR #508: Adds `AmberToolsToolkitWrapper.assign_fractional_bond_orders`.
- PR #469: The `Molecule` class adds `Molecule.has_unique_atom_names` and `Molecule.has_unique_atom_names`.
- PR #472: Adds to the `Molecule` class `Molecule.are_isomorphic` and `Molecule.is_isomorphic_with` and `Molecule.hill_formula` and `Molecule.to_hill_formula` and `Molecule.to_qcschema` and `Molecule.from_qcschema` and `Molecule.from_mapped_smiles` and `Molecule.from_pdb_and_smiles` and `Molecule.canonical_order_atoms` and `Molecule.remap`

  Note: The to_qcschema method accepts an extras dictionary which is passed into the validated qcelemental.models.Molecule object.

- PR #506: The `Molecule` class adds `Molecule.find_rotatable_bonds`
- PR #521: Adds `Molecule.to_inchi` and `Molecule.to_inchikey` and `Molecule.from_inchi`

  Warning: InChI was not designed as an molecule interchange format and using it as one is not recommended. Many round trip tests will fail when using this format due to a loss of information. We have also added support for fixed hydrogen layer nonstandard InChI which can help in the case of tautomers, but overall creating molecules from InChI should be avoided.

- PR #529: Adds the ability to write out to XYZ files via `Molecule.to_file` Both single frame and multiframe XYZ files are supported. Note reading from XYZ files will not be supported due to the lack of connectivity information.
- PR #535: Extends the the API for the `Molecule.to_smiles` to allow for the creation of cmiles identifiers through combinations of isomeric, explicit hydrogen and mapped smiles, the default settings will return isomeric explicit hydrogen smiles as expected.

  Warning: Atom maps can be supplied to the properties dictionary to modify which atoms have their map index included, if no map is supplied all atoms will be mapped in the order they appear in the `Molecule`.

- PR #563: Adds `test_forcefields/ion_charges.offxml`, giving `LibraryCharges` for monatomic ions.
- PR #543: Adds 3 new methods to the `Molecule` class which allow the enumeration of molecule

states. These are `Molecule.enumerate_tautomers`, `Molecule.enumerate_stereoisomers`, `Molecule.enumerate_protomers`

> **Warning:** Enumerate protomers is currently only available through the OpenEye toolkit.

- PR #573: Adds quacpac error output to quacpac failure in `Molecule.compute_partial_charges_am1bcc`.
- PR #560: Added visualization method to the the Molecule class.
- PR #620: Added the ability to register parameter handlers via entry point plugins. This functionality is accessible by initializing a `ForceField` with the `load_plugins=True` keyword argument.
- PR #582: Added fractional bond order interpolation Adds *return_topology* kwarg to `Forcefield.create_openmm_system`, which returns the processed topology along with the OpenMM `System` when `True` (default `False`).

Tests added

- PR #558: Adds tests ensuring that the new Topology particle indexing system are properly implemented, and that TopologyVirtualSites reference the correct TopologyAtoms.
- PR #469: Added round-trip SMILES test to add coverage for `Molecule.from_smiles`.
- PR #469: Added tests for unique atom naming behavior in `Topology.to_openmm`, as well as tests of the `ensure_unique_atom_names=False` kwarg disabling this behavior.
- PR #472: Added tests for `Molecule.hill_formula` and `Molecule.to_hill_formula` for the various supported input types.
- PR #472: Added round-trip test for `Molecule.from_qcschema` and `Molecule.to_qcschema`.
- PR #472: Added tests for `Molecule.is_isomorphic_with` and `Molecule.are_isomorphic` with various levels of isomorphic graph matching.
- PR #472: Added toolkit dependent tests for `Molecule.canonical_order_atoms` due to differences in the algorithms used.
- PR #472: Added a test for `Molecule.from_mapped_smiles` using the molecule from issue #412 to ensure it is now fixed.
- PR #472: Added a test for `Molecule.remap`, this also checks for expected error when the mapping is not complete.
- PR #472: Added tests for `Molecule.from_pdb_and_smiles` to check for a correct combination of smiles and PDB and incorrect combinations.
- PR #509: Added test for `Molecule.chemical_environment_matches` to check that the complete set of matches is returned.
- PR #509: Added test for `Forcefield.create_openmm_system` to check that a protein system can be created.
- PR #506: Added a test for the molecule identified in issue #513 as losing aromaticity when converted to rdkit.
- PR #506: Added a verity of toolkit dependent tests for identifying rotatable bonds while ignoring the user requested types.

- PR #521: Added toolkit independent round-trip InChI tests which add coverage for `Molecule.to_inchi` and `Molecule.from_inchi`. Also added coverage for bad inputs and `Molecule.to_inchikey`.
- PR #529: Added to XYZ file coverage tests.
- PR #563: Added *LibraryCharges* parameterization test for monatomic ions in `test_forcefields/ion_charges.offxml`.
- PR #543: Added tests to assure that state enumeration can correctly find molecules tautomers, stereoisomers and protomers when possible.
- PR #573: Added test for `quacpac` error output for `quacpac` failure in `Molecule.compute_partial_charges_am1bcc`.
- PR #579: Adds regression tests to ensure RDKit can be be used to write multi-model PDB files.
- PR #582: Added fractional bond order interpolation tests, tests for `ValidatedDict`.

Bugfixes

- PR #558: Fixes a bug where `TopologyVirtualSite.atoms` would not correctly apply `TopologyMolecule` atom ordering on top of the reference molecule ordering, in cases where the same molecule appears multiple times, but in a different order, in the same Topology.
- Issue #460: Creates unique atom names in `Topology.to_openmm` if the existing ones are not unique. The lack of unique atom names had been causing problems in workflows involving downstream tools that expect unique atom names.
- Issue #448: We can now make molecules from mapped smiles using `Molecule.from_mapped_smiles` where the order will correspond to the indeing used in the smiles. Molecules can also be re-indexed at any time using the `Molecule.remap`.
- Issue #462: We can now instance the `Molecule` from a QCArchive entry record instance or dictionary representation.
- Issue #412: We can now instance the `Molecule` using `Molecule.from_mapped_smiles`. This resolves an issue caused by RDKit considering atom map indices to be a distinguishing feature of an atom, which led to erroneous definition of chirality (as otherwise symmetric substituents would be seen as different). We anticipate that this will reduce the number of times you need to type `allow_undefined_stereo=True` when processing molecules that do not actually contain stereochemistrty.
- Issue #513: The `Molecule.to_rdkit` now re-sets the aromaticity model after sanitizing the molecule.
- Issue #500: The `Molecule.find_rotatable_bonds` has been added which returns a list of rotatable `Bond` instances for the molecule.
- Issue #491: We can now parse large molecules without hitting a match limit cap.
- Issue #474: We can now convert molecules to InChI and InChIKey and from InChI.
- Issue #523: The `Molecule.to_file` method can now correctly write to `MOL` files, in line with the supported file type list.
- Issue #568: The `Molecule.to_file` can now correctly write multi-model PDB files when using the RDKit backend toolkit.

Examples added

- PR #591 and PR #533: Adds an example notebook and utility to compute conformer energies. This example is made to be reverse-compatible with the 0.6.0 OpenFF Toolkit release.
- PR #472: Adds an example notebook QCarchive_interface.ipynb which shows users how to instance the `Molecule` from a QCArchive entry level record and calculate the energy using RDKit through QCEngine.

0.6.0 - Library Charges

This release adds support for a new SMIRKS-based charge assignment method, Library Charges. The addition of more charge assignment methods opens the door for new types of experimentation, but also introduces several complex behaviors and failure modes. Accordingly, we have made changes to the charge assignment infrastructure to check for cases when partial charges do not sum to the formal charge of the molecule, or when no charge assignment method is able to generate charges for a molecule. More detailed explanation of the new errors that may be raised and keywords for overriding them are in the "Behavior Changed" section below.

With this release, we update `test_forcefields/tip3p.offxml` to be a working example of assigning LibraryCharges. However, we do not provide any force field files to assign protein residue `LibraryCharges`. If you are interested in translating an existing protein FF to SMIRNOFF format or developing a new one, please feel free to contact us on the Issue tracker or open a Pull Request.

New features

- PR #433: Closes Issue #25 by adding initial support for the LibraryCharges tag in the SMIRNOFF specification using `LibraryChargeHandler`. For a molecule to have charges assigned using Library Charges, all of its atoms must be covered by at least one `LibraryCharge`. If an atom is covered by multiple `LibraryCharge` s, then the last `LibraryCharge` matched will be applied (per the hierarchy rules in the SMIRNOFF format).

  This functionality is thus able to apply per-residue charges similar to those in traditional protein force fields. At this time, there is no concept of "residues" or "fragments" during parametrization, so it is not possible to assign charges to *some* atoms in a molecule using `LibraryCharge` s, but calculate charges for other atoms in the same molecule using a different method. To assign charges to a protein, LibraryCharges SMARTS must be provided for the residues and protonation states in the molecule, as well as for any capping groups and post-translational modifications that are present.

  It is valid for `LibraryCharge` SMARTS to *partially* overlap one another. For example, a molecule consisting of atoms `A-B-C` connected by single bonds could be matched by a SMIRNOFF `LibraryCharges` section containing two `LibraryCharge` SMARTS: `A-B` and `B-C`. If listed in that order, the molecule would be assigned the `A` charge from the `A-B` `LibraryCharge` element and the `B` and `C` charges from the `B-C` element. In testing, these types of partial overlaps were found to frequently be sources of undesired behavior, so it is recommended that users define whole-molecule `LibraryCharge` SMARTS whenever possible.
- PR #455: Addresses Issue #393 by adding `ParameterHandler.attribute_is_cosmetic` and `ParameterType.attribute_is_cosmetic`, which return True if the provided attribute name is defined for the queried object but does not correspond to an allowed value in the SMIRNOFF spec.

Behavior changed

- PR #433: If a molecule can not be assigned charges by any charge-assignment method, an `openforcefield.typing.engines.smirnoff.parameters.UnassignedMoleculeChargeException` will be raised. Previously, creating a system without either `ToolkitAM1BCCHandler` or the `charge_from_molecules` keyword argument to `ForceField.create_openmm_system` would produce an OpenMM `System` where the molecule has zero charge on all atoms. However, given that we will soon be adding more options for charge assignment, it is important that failures not be silent. Molecules with zero charge can still be produced by setting the `Molecule.partial_charges` array to be all zeroes, and including the molecule in the `charge_from_molecules` keyword argument to `create_openmm_system`.
- PR #433: Due to risks introduced by permitting charge assignment using partially-overlapping `LibraryCharge` s, the toolkit will now raise a `openforcefield.typing.engines.smirnoff.parameters.NonIntegralMoleculeChargeException` if the sum of partial charges on a molecule are found to be more than 0.01 elementary charge units different than the molecule's formal charge. This exception can be overridden by providing the `allow_nonintegral_charges=True` keyword argument to `ForceField.create_openmm_system`.

Tests added

- PR #430: Added test for Wiberg Bond Order implemented in OpenEye Toolkits. Molecules taken from DOI:10.5281/zenodo.3405489 . Added by Sukanya Sasmal.
- PR #569: Added round-trip tests for more serialization formats (dict, YAML, TOML, JSON, BSON, messagepack, pickle). Note that some are unsupported, but the tests raise the appropriate error.

Bugfixes

- PR #431: Fixes an issue where `ToolkitWrapper` objects would improperly search for functionality in the `GLOBAL_TOOLKIT_REGISTRY`, even though a specific `ToolkitRegistry` was requested for an operation.
- PR #439: Fixes Issue #438, by replacing call to NetworkX `Graph.node` with call to `Graph.nodes`, per 2.4 migration guide.

Files modified

- PR #433: Updates the previously-nonfunctional `test_forcefields/tip3p.offxml` to a functional state by updating it to the SMIRNOFF 0.3 specification, and specifying atomic charges using the `LibraryCharges` tag.

0.5.1 - Adding the parameter coverage example notebook

This release contains a new notebook example, check_parameter_coverage.ipynb, which loads sets of molecules, checks whether they are parameterizable, and generates reports of chemical motifs that are not. It also fixes several simple issues, improves warnings and docstring text, and removes unused files.

The parameter coverage example notebook goes hand-in-hand with the release candidate of our initial force field, openff-1.0.0-RC1.offxml , which will be temporarily available until the official force field release is made in October. Our goal in publishing this notebook alongside our first major refitting is to allow interested users to check whether there is parameter coverage for their molecules of interest. If the force field is unable to parameterize a molecule, this notebook will generate reports of the specific chemistry that is not covered.

We understand that many organizations in our field have restrictions about sharing specific molecules, and the outputs from this notebook can easily be cropped to communicate unparameterizable chemistry without revealing the full structure.

The force field release candidate is in our new refit force field package, openforcefields. This package is now a part of the Open Force Field Toolkit conda recipe, along with the original smirnoff99Frosst line of force fields.

Once the `openforcefields` conda package is installed, you can load the release candidate using:

```
ff = ForceField('openff-1.0.0-RC1.offxml')
```

The release candidate will be removed when the official force field, `openff-1.0.0.offxml`, is released in early October.

Complete details about this release are below.

Example added

- PR #419: Adds an example notebook check_parameter_coverage.ipynb which shows how to use the toolkit to check a molecule dataset for missing parameter coverage, and provides functionality to output tagged SMILES and 2D drawings of the unparameterizable chemistry.

New features

- PR #419: Unassigned valence parameter exceptions now include a list of tuples of `TopologyAtom` which were unable to be parameterized (`exception.unassigned_topology_atom_tuples`) and the class of the `ParameterHandler` that raised the exception (`exception.handler_class`).
- PR #425: Implements Trevor Gokey's suggestion from Issue #411, which enables pickling of `ForceFields` and `ParameterHandlers`. Note that, while XML representations of `ForceFields` are stable and conform to the SMIRNOFF specification, the pickled `ForceFields` that this functionality enables are not guaranteed to be compatible with future toolkit versions.

Improved documentation and warnings

- PR #425: Addresses Issue #410, by explicitly having toolkit warnings print `Warning:` at the beginning of each warning, and adding clearer language to the warning produced when the OpenEye Toolkits can not be loaded.
- PR #425: Addresses Issue #421 by adding type/shape information to all Molecule partial charge and conformer docstrings.
- PR #425: Addresses Issue #407 by providing a more extensive explanation of why we don't use RDKit's mol2 parser for molecule input.

Bugfixes

- PR #419: Fixes Issue #417 and Issue #418, where `RDKitToolkitWrapper.from_file` would disregard the `allow_undefined_stereo` kwarg and skip the first molecule when reading a SMILES file.

Files removed

- PR #425: Addresses Issue #424 by deleting the unused files `openforcefield/typing/engines/smirnoff/gbsaforces.py` and `openforcefield/tests/test_smirnoff.py`. `gbsaforces.py` was only used internally and `test_smirnoff.py` tested unsupported functionality from before the 0.2.0 release.

0.5.0 - GBSA support and quality-of-life improvements

This release adds support for the GBSA tag in the SMIRNOFF specification. Currently, the `HCT`, `OBC1`, and `OBC2` models (corresponding to AMBER keywords `igb=1`, `2`, and `5`, respectively) are supported, with the `OBC2` implementation being the most flexible. Unfortunately, systems produced using these keywords are not yet transferable to other simulation packages via ParmEd, so users are restricted to using OpenMM to simulate systems with GBSA.

OFFXML files containing GBSA parameter definitions are available, and can be loaded in addition to existing parameter sets (for example, with the command `ForceField('test_forcefields/smirnoff99Frosst.offxml', 'test_forcefields/GBSA_OBC1-1.0.offxml')`). A manifest of new SMIRNOFF-format GBSA files is below.

Several other user-facing improvements have been added, including easier access to indexed attributes, which are now accessible as `torsion.k1`, `torsion.k2`, etc. (the previous access method `torsion.k` still works as well). More details of the new features and several bugfixes are listed below.

New features

- PR #363: Implements `GBSAHandler`, which supports the GBSA tag in the SMIRNOFF specification. Currently, only GBSAHandlers with `gb_model="OBC2"` support setting non-default values for the `surface_area_penalty` term (default `5.4*calories/mole/angstroms**2`), though users can zero the SA term for `OBC1` and `HCT` models by setting `sa_model="None"`. No model currently supports setting `solvent_radius` to any value other than `1.4*angstroms`. Files containing experimental SMIRNOFF-format implementations of `HCT`, `OBC1`, and `OBC2` are included with this release (see below). Additional details of these models, including literature references, are available on the SMIRNOFF specification page.

> **Warning:** The current release of ParmEd can not transfer GBSA models produced by the Open Force Field Toolkit to other simulation packages. These GBSA forces are currently only computable using OpenMM.

- PR #363: When using `Topology.to_openmm()`, periodic box vectors are now transferred from the Open Force Field Toolkit Topology into the newly-created OpenMM Topology.
- PR #377: Single indexed parameters in `ParameterHandler` and `ParameterType` can now be get/set through normal attribute syntax in addition to the list syntax.
- PR #394: Include element and atom name in error output when there are missing valence parameters during molecule parameterization.

Bugfixes

- PR #385: Fixes Issue #346 by having `OpenEyeToolkitWrapper.compute_partial_charges_am1bcc` fall back to using standard AM1-BCC if AM1-BCC ELF10 charge generation raises an error about "trans COOH conformers"
- PR #399: Fixes issue where `ForceField` constructor would ignore `parameter_handler_classes` kwarg.
- PR #400: Makes link-checking tests retry three times before failing.

Files added

- PR #363: Adds `test_forcefields/GBSA_HCT-1.0.offxml`, `test_forcefields/GBSA_OBC1-1.0.offxml`, and `test_forcefields/GBSA_OBC2-1.0.offxml`, which are experimental implementations of GBSA models. These are primarily used in validation tests against OpenMM's models, and their version numbers will increment if bugfixes are necessary.

0.4.1 - Bugfix Release

This update fixes several toolkit bugs that have been reported by the community. Details of these bugfixes are provided below.

It also refactors how `ParameterType` and `ParameterHandler` store their attributes, by introducing `ParameterAttribute` and `IndexedParameterAttribute`. These new attribute-handling classes provide a consistent backend which should simplify manipulation of parameters and implementation of new handlers.

Bug fixes

- PR #329: Fixed a bug where the two `BondType` parameter attributes `k` and `length` were treated as indexed attributes. (`k` and `length` values that correspond to specific bond orders will be indexed under `k_bondorder1`, `k_bondorder2`, etc when implemented in the future)
- PR #329: Fixed a bug that allowed setting indexed attributes to single values instead of strictly lists.
- PR #370: Fixed a bug in the API where `BondHandler`, `ProperTorsionHandler` , and `ImproperTorsionHandler` exposed non-functional indexed parameters.
- PR #351: Fixes Issue #344, in which the main `FrozenMolecule` constructor and several other Molecule-construction functions ignored or did not expose the `allow_undefined_stereo` keyword argument.
- PR #351: Fixes a bug where a molecule which previously generated a SMILES using one cheminformatics toolkit returns the same SMILES, even though a different toolkit (which would generate a different SMILES for the molecule) is explicitly called.
- PR #354: Fixes the error message that is printed if an unexpected parameter attribute is found while loading data into a `ForceField` (now instructs users to specify `allow_cosmetic_attributes` instead of `permit_cosmetic_attributes`)
- PR #364: Fixes Issue #362 by modifying `OpenEyeToolkitWrapper.from_smiles` and `RDKitToolkitWrapper.from_smiles` to make implicit hydrogens explicit before molecule creation. These functions also now raise an error if the optional keyword `hydrogens_are_explicit=True` but the SMILES are interpreted by the backend cheminformatic toolkit as having implicit hydrogens.
- PR #371: Fixes error when reading early SMIRNOFF 0.1 spec files enclosed by a top-level `SMIRFF` tag.

Note: The enclosing `SMIRFF` tag is present only in legacy files. Since developing a formal specification, the only acceptable top-level tag value in a SMIRNOFF data structure is `SMIRNOFF`.

Code enhancements

- PR #329: `ParameterType` was refactored to improve its extensibility. It is now possible to create new parameter types by using the new descriptors `ParameterAttribute` and `IndexedParameterAttribute`.
- PR #357: Addresses Issue #356 by raising an informative error message if a user attempts to load an OpenMM topology which is probably missing connectivity information.

Force fields added

- PR #368: Temporarily adds `test_forcefields/smirnoff99frosst_experimental.offxml` to address hierarchy problems, redundancies, SMIRKS pattern typos etc., as documented in issue #367. Will ultimately be propagated to an updated force field in the `openforcefield/smirnoff99frosst` repo.
- PR #371: Adds `test_forcefields/smirff99Frosst_reference_0_1_spec.offxml`, a SMIRNOFF 0.1 spec file enclosed by the legacy `SMIRFF` tag. This file is used in backwards-compatibility testing.

0.4.0 - Performance optimizations and support for SMIRNOFF 0.3 specification

This update contains performance enhancements that significantly reduce the time to create OpenMM systems for topologies containing many molecules via `ForceField.create_openmm_system`.

This update also introduces the SMIRNOFF 0.3 specification. The spec update is the result of discussions about how to handle the evolution of data and parameter types as further functional forms are added to the SMIRNOFF spec.

We provide methods to convert SMIRNOFF 0.1 and 0.2 force fields written with the XML serialization (`.offxml`) to the SMIRNOFF 0.3 specification. These methods are called automatically when loading a serialized SMIRNOFF data representation written in the 0.1 or 0.2 specification. This functionality allows the toolkit to continue to read files containing SMIRNOFF 0.2 spec force fields, and also implements backwards-compatibility for SMIRNOFF 0.1 spec force fields.

Warning: The SMIRNOFF 0.1 spec did not contain fields for several energy-determining parameters that are exposed in later SMIRNOFF specs. Thus, when reading SMIRNOFF 0.1 spec data, the toolkit must make assumptions about the values that should be added for the newly-required fields. The values that are added include 1-2, 1-3 and 1-5 scaling factors, cutoffs, and long-range treatments for nonbonded interactions. Each assumption is printed as a warning during the conversion process. Please carefully review the warning messages to ensure that the conversion is providing your desired behavior.

SMIRNOFF 0.3 specification updates

- The SMIRNOFF 0.3 spec introduces versioning for each individual parameter section, allowing asynchronous updates to the features of each parameter class. The top-level `SMIRNOFF` tag, containing information like `aromaticity_model`, `Author`, and `Date`, still has a version (currently 0.3). But, to allow for independent development of individual parameter types, each section (such as `Bonds`, `Angles`, etc) now has its own version as well (currently all 0.3).
- All units are now stored in expressions with their corresponding values. For example, distances are now stored as `1.526*angstrom`, instead of storing the unit separately in the section header.
- The current allowed value of the `potential` field for `ProperTorsions` and `ImproperTorsions` tags is no longer `charmm`, but is rather `k*(1+cos(periodicity*theta-phase))`. It was pointed out to us that CHARMM-style torsions deviate from this formula when the periodicity of a torsion term is 0, and we do not intend to reproduce that behavior.
- SMIRNOFF spec documentation has been updated with tables of keywords and their defaults for each parameter section and parameter type. These tables will track the allowed keywords and default behavior as updated versions of individual parameter sections are released.

Performance improvements and bugfixes

- PR #329: Performance improvements when creating systems for topologies with many atoms.
- PR #347: Fixes bug in charge assignment that occurs when charges are read from file, and reference and charge molecules have different atom orderings.

New features

- PR #311: Several new experimental functions.
  - Adds `convert_0_2_smirnoff_to_0_3`, which takes a SMIRNOFF 0.2-spec data dict, and updates it to 0.3. This function is called automatically when creating a `ForceField` from a SMIRNOFF 0.2 spec OFFXML file.
  - Adds `convert_0_1_smirnoff_to_0_2`, which takes a SMIRNOFF 0.1-spec data dict, and updates it to 0.2. This function is called automatically when creating a `ForceField` from a SMIRNOFF 0.1 spec OFFXML file.
  - NOTE: The format of the "SMIRNOFF data dict" above is likely to change significantly in the future. Users that require a stable serialized ForceField object should use the output of `ForceField.to_string('XML')` instead.
  - Adds `ParameterHandler` and `ParameterType` `add_cosmetic_attribute` and `delete_cosmetic_attribute` functions. Once created, cosmetic attributes can be accessed and modified as attributes of the underlying object (eg. `ParameterType.my_cosmetic_attrib = 'blue'`) These functions are experimental, and we are interested in feedback on how cosmetic attribute handling could be improved. (See Issue #338) Note that if a new cosmetic attribute is added to an object without using these functions, it will not be recognized by the toolkit and will not be written out during serialization.
  - Values for the top-level `Author` and `Date` tags are now kept during SMIRNOFF data I/O. If multiple data sources containing these fields are read, the values are concatenated using "AND" as a separator.

API-breaking changes

- `ForceField.to_string` and `ForceField.to_file` have had the default value of their `discard_cosmetic_attributes` kwarg set to False.
- `ParameterHandler` and `ParameterType` constructors now expect the `version` kwarg (per the SMIRNOFF spec change above) This requirement can be skipped by providing the kwarg `skip_version_check=True`
- `ParameterHandler` and `ParameterType` functions no longer handle `X_unit` attributes in SMIRNOFF data (per the SMIRNOFF spec change above).
- The scripts in `utilities/convert_frosst` are now deprecated. This functionality is important for provenance and will be migrated to the `openforcefield/smirnoff99Frosst` repository in the coming weeks.
- `ParameterType._SMIRNOFF_ATTRIBS` is now `ParameterType._REQUIRED_SPEC_ATTRIBS`, to better parallel the structure of the `ParameterHandler` class.
- `ParameterType._OPTIONAL_ATTRIBS` is now `ParameterType._OPTIONAL_SPEC_ATTRIBS`, to better parallel the structure of the `ParameterHandler` class.
- Added class-level dictionaries `ParameterHandler._DEFAULT_SPEC_ATTRIBS` and `ParameterType._DEFAULT_SPEC_ATTRIBS`.

0.3.0 - API Improvements

Several improvements and changes to public API.

New features

- PR #292: Implement `Topology.to_openmm` and remove `ToolkitRegistry.toolkit_is_available`
- PR #322: Install directories for the lookup of OFFXML files through the entry point group `openforcefield.smirnoff_forcefield_directory`. The `ForceField` class doesn't search in the `data/forcefield/` folder anymore (now renamed `data/test_forcefields/`), but only in `data/`.

API-breaking Changes

- PR #278: Standardize variable/method names
- PR #291: Remove `ForceField.load/to_smirnoff_data`, add `ForceField.to_file/string` and `ParameterHandler.add_parameters`. Change behavior of `ForceField.register_X_handler` functions.

Bugfixes

- PR #327: Fix units in tip3p.offxml (note that this file is still not loadable by current toolkit)
- PR #325: Fix solvent box for provided test system to resolve periodic clashes.
- PR #325: Add informative message containing Hill formula when a molecule can't be matched in `Topology.from_openmm`.
- PR #325: Provide warning or error message as appropriate when a molecule is missing stereochemistry.
- PR #316: Fix formatting issues in GBSA section of SMIRNOFF spec

- PR #308: Cache molecule SMILES to improve system creation speed
- PR #306: Allow single-atom molecules with all zero coordinates to be converted to OE/RDK mols
- PR #313: Fix issue where constraints are applied twice to constrained bonds

0.2.2 - Bugfix release

This release modifies an example to show how to parameterize a solvated system, cleans up backend code, and makes several improvements to the README.

Bugfixes

- PR #279: Cleanup of unused code/warnings in main package `__init__`
- PR #259: Update T4 Lysozyme + toluene example to show how to set up solvated systems
- PR #256 and PR #274: Add functionality to ensure that links in READMEs resolve successfully

0.2.1 - Bugfix release

This release features various documentation fixes, minor bugfixes, and code cleanup.

Bugfixes

- PR #267: Add neglected `<ToolkitAM1BCC>` documentation to the SMIRNOFF 0.2 spec
- PR #258: General cleanup and removal of unused/inaccessible code.
- PR #244: Improvements and typo fixes for BRD4:inhibitor benchmark

0.2.0 - Initial RDKit support

This version of the toolkit introduces many new features on the way to a 1.0.0 release.

New features

- Major overhaul, resulting in the creation of the SMIRNOFF 0.2 specification and its XML representation
- Updated API and infrastructure for reference SMIRNOFF `ForceField` implementation
- Implementation of modular `ParameterHandler` classes which process the topology to add all necessary forces to the system.
- Implementation of modular `ParameterIOHandler` classes for reading/writing different serialized SMIRNOFF force field representations
- Introduction of `Molecule` and `Topology` classes for representing molecules and biomolecular systems
- New `ToolkitWrapper` interface to RDKit, OpenEye, and AmberTools toolkits, managed by `ToolkitRegistry`
- API improvements to more closely follow PEP8 guidelines
- Improved documentation and examples

0.1.0

This is an early preview release of the toolkit that matches the functionality described in the preprint describing the SMIRNOFF v0.1 force field format: [DOI].

New features

This release features additional documentation, code comments, and support for automated testing.

Bugfixes

Treatment of improper torsions

A significant (though currently unused) problem in handling of improper torsions was corrected. Previously, non-planar impropers did not behave correctly, as six-fold impropers have two potential chiralities. To remedy this, SMIRNOFF impropers are now implemented as three-fold impropers with consistent chirality. However, current force fields in the SMIRNOFF format had no non-planar impropers, so this change is mainly aimed at future work.

CHAPTER

FOUR

FREQUENTLY ASKED QUESTIONS (FAQ)

4.1 What kinds of input files can I apply SMIRNOFF parameters to?

SMIRNOFF force fields use direct chemical perception meaning that, unlike many molecular mechanics (MM) force fields, they apply parameters based on substructure searches acting directly on molecules. This creates unique opportunities and allows them to encode a great deal of chemistry quite simply, but it also means that the *starting point* for parameter assignment must be well-defined chemically, giving not just the elements and connectivity for all of the atoms of all of the components of your system, but also providing the formal charges and bond orders.

Specifically, to apply SMIRNOFF to a system, you must either:

1. Provide Open Force Field Toolkit `Molecule` objects corresponding to the components of your system, or
2. Provide an OpenMM `Topology` which includes bond orders and thus can be converted to molecules corresponding to the components of your system

Without this information, our direct chemical perception cannot be applied to your molecule, as it requires the chemical identity of the molecules in your system – that is, bond order and formal charge as well as atoms and connectivity. Unless you provide the full chemical identity in this sense, we must attempt to guess or infer the chemical identity of your molecules, which is a recipe for trouble. Different molecules can have the same chemical graph but differ in bond order and formal charge, or different resonance structures may be treated rather differently by some force fields (e.g. `c1cc(ccc1c2cc[nH+]cc2)[O-]` vs `C1=CC(C=CC1=C2C=CNC=C2)=O`, where the central bond is rotatable in one resonance structure but not in the other) even though they have identical formal charge and connectivity (chemical graph). A force field which uses the chemical identity of molecules to assign parameters needs to know the exact chemical identity of the molecule you are intending to parameterize.

4.2 Can I use an AMBER (or GROMACS) topology/coordinate file as a starting point for applying a SMIRNOFF force field?

In a word, “no”.

Parameter files used by typical molecular dynamics simulation packages do not currently encode enough information to identify the molecules chemically present, or at least not without drawing inferences. For example, one could take a structure file and infer bond orders based on bond lengths, or attempt to infer bond orders from force constants in a parameter file. Such inference work is outside the scope of SMIRNOFF.

If you have such an inference problem, we recommend that you use pre-existing cheminformatics tools available elsewhere (such as via the OpenEye toolkits, such as the `OEPerceiveBondOrders` functionality offered there) to solve this problem and identify your molecules before beginning your work with SMIRNOFF.

4.3 What about starting from a PDB file?

PDB files do not in general provide the chemical identity of small molecules contained therein, and thus do not provide suitable starting points for applying SMIROFF to small molecules. This is especially problematic for PDB files from X-ray crystallography which typically do not include proteins, making the problem even worse. For our purposes here, however, we assume you begin with the coordinates of all atoms present and the full topology of your system.

Given a PDB file of a hypothetical biomolecular system of interest containing a small molecule, there are several routes available to you for treating the small molecule present:

- Use a cheminformatics toolkit (see above) to infer bond orders
- Identify your ligand from a database; e.g. if it is in the Protein Data Bank (PDB), it will be present in the Ligand Expo meaning that it has a database entry and code you can use to look up its putative chemical identity
- Identify your ligand by name or SMILES string (or similar) from the literature or your collaborators

4.4 What do you recommend as a starting point?

For application of SMIRNOFF force fields, we recommend that you begin your work with formats which provide the chemical identity of your small molecule (including formal charge and bond order). This means we recommend one of the following or equivalent:

- A `.mol2` file or files for the molecules comprising your system, with correct bond orders and formal charges. (Note: Do NOT generate this from a simulation package or tool which does not have access to bond order information; you may end up with a `.mol2` file, but the bond orders will be incorrect)
- Isomeric SMILES strings for the components of your system
- InCHI strings for the components of your system
- Chemical Identity Registry numbers for the components of your system
- IUPAC names for the components of your system

Essentially, anything which provides the full identity of what you want to simulate (including stereochemistry) should work, though it may require more or less work to get it into an acceptable format.

4.5 My conda installation of the toolkit doesn't appear to work. What should I try next?

We recommend that you install the toolkit in a fresh conda environment, explicitly passing the channels to be used, in-order:

```
conda create -n <my_new_env> -c conda-forge openff-toolkit
conda activate <my_new_env>
```

Installing into a new environment avoids forcing conda to satisfy the dependencies of both the toolkit and all existing packages in that environment. Taking the approach that conda environments are generally disposable, even ephemeral, minimizes the chances for hard-to-diagnose dependency issues.

4.6 The partial charges generated by the toolkit don't seem to depend on the molecule's conformation! Is this a bug?

No! This is the intended behavior. The force field parameters of a molecule should be independent of both their chemical environment and conformation so that they can be used and compared across different contexts. When applying AM1BCC partial charges, the toolkit achieves a deterministic output by ignoring the input conformation and producing several new conformations for the same molecule. Partial charges are then computed based on these conformations. This behavior can be controlled with the `use_conformers` argument to the assign_partial_charges() and compute_partial_charges_am1bcc() methods of the Molecule class.

4.7 How can I distribute my own force fields in SMIRNOFF format?

We support conda data packages for distribution of force fields in `.offxml` format! Just add the relevant entry point to `setup.py` and distribute on conda Forge:

```
entry_points={
    'openforcefield.smirnoff_forcefield_directory' : [
        'my_new_force_field_paths = my_package:get_my_new_force_field_paths',
    ],
}
```

Where `get_my_new_force_field_paths` is a function in the `my_package` module providing a list of strings holding the paths to the directories to search. You should also rename `my_new_force_field_paths` to suit your force field. See `openff-forcefields` for an example.

CHAPTER

FIVE

CORE CONCEPTS

OpenFF `Molecule` A graph representation of a molecule containing enough information to unambiguously parametrize it. Required data fields for a `Molecule` are:

- `atoms`: element (integer), formal_charge (integer), is_aromatic (boolean), stereochemistry (`'R'/'S'/None`)
- `bonds`: order (integer), is_aromatic (boolean), stereochemistry (`'E'/'Z'/None`)

There are several other optional attributes such as `conformers` and `partial_charges` that may be populated in the `Molecule` data structure. These are considered "optional" because they are not required for system creation, however if those fields are populated, the user MAY use them to override values that would otherwise be generated during system creation.

A dictionary, `Molecule.properties` is exposed, which is a Python dict that can be populated with arbitrary data. This data should be considered cosmetic and should not affect system creation. Whenever possible, molecule serialization or format conversion should preserve this data.

OpenFF `System` An object that contains everything needed to calculate a molecular system's energy, except the atomic coordinates. Note that this does not exist yet, and that OpenMM `System` objects are being used for this purpose right now. Development is underway on GitHub.

OpenFF `Topology` An object that efficiently holds many OpenFF `Molecule` objects. The atom indexing in a `Topology` may differ from those of the underlying `Molecules`

OpenFF `TopologyMolecule` The efficient data structures that make up an OpenFF `Topology`. There is one `TopologyMolecule` for each instance of a chemical species in a `Topology`. However, each unique chemical species has a single OpenFF `Molecule` representing it, which may be shared by multiple `TopologyMolecules`. `TopologyMolecules` contain an atom index map, as several copies of the same chemical species in a `Topology` may be present with different atom orderings. This data structure allows the OpenFF toolkit to only parametrize each unique `Molecule` once, and then write a copy of the assigned parameters out for each of the `Molecule` in the `Topology` (accounting for atom indexing differences in the process).

OpenFF `ForceField` An object generated from an OFFXML file (or other source of SMIRNOFF data). Most information from the SMIRNOFF data source is stored in this object's several `ParameterHandlers`, however some top-level SMIRNOFF data is stored in the `ForceField` object itself.

CHAPTER

SIX

COOKBOOK: EVERY WAY TO MAKE A `MOLECULE`

Every pathway through the OpenFF Toolkit boils down to four steps:

1. Using other tools, assemble a graph of a molecule, including all of its atoms, bonds, bond orders, formal charges, and stereochemistry[1]
2. Use that information to construct a `Molecule`
3. Combine a number of `Molecule` objects to construct a `Topology`
4. Call `ForceField.create_openmm_system(topology)` to create an OpenMM `System` (or, in the near future, an OpenFF `Interchange` for painless conversion to all sorts of MD formats)

So let's take a look at every way there is to construct a molecule! We'll use zwitterionic L-alanine as an example biomolecule with all the tricky bits - a stereocenter, non-zero formal charges, and bonds of different orders.

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_280/4122961659.py in <module>
      1 # Imports
----> 2 from openff.toolkit.topology import Molecule, Topology
      3 from openff.toolkit.typing.engines.smirnoff import ForceField
      4
      5 # Hide tracebacks for simpler errors

~/checkouts/readthedocs.org/user_builds/open-forcefield-toolkit/conda/topology/lib/python3.7/
↪site-packages/openff_toolkit-0.10.0+75.g7b1a97f3.dirty-py3.7.egg/openff/toolkit/topology/__
↪init__.py in <module>
----> 1 from openff.toolkit.topology.molecule import (
      2     Atom,
      3     Bond,
      4     BondChargeVirtualSite,
      5     DivalentLonePairVirtualSite,

~/checkouts/readthedocs.org/user_builds/open-forcefield-toolkit/conda/topology/lib/python3.7/
↪site-packages/openff_toolkit-0.10.0+75.g7b1a97f3.dirty-py3.7.egg/openff/toolkit/topology/
↪molecule.py in <module>
     41 import networkx as nx
     42 import numpy as np
---> 43 from cached_property import cached_property
```

(continues on next page)

[1] Note that this stereochemistry must be defined on the *graph* of the molecule. It's not good enough to just co-ordinates with the correct stereochemistry. But if you have the co-ordinates, you can try getting the stereochemistry automatically with `rdkit` or `openeye` — If you dare!

(continued from previous page)

```
     44 from packaging import version
     45

ModuleNotFoundError: No module named 'cached_property'
```

6.1 From SMILES

SMILES is the classic way to create a `Molecule`. SMILES is a widely-used compact textual representation of arbitrary molecules. This lets us specify an exact molecule, including stereochemistry and bond orders, very easily — though they may not be the most human-readable format.

The `Molecule.from_smiles()` method is used to create a `Molecule` from a SMILES code.

6.1.1 Implicit hydrogens SMILES

```
zw_l_alanine = Molecule.from_smiles("C[C@H]([NH3+])C(=O)[O-]")

zw_l_alanine.visualize()
```

6.1.2 Explicit hydrogens SMILES

```
smiles_explicit_h = Molecule.from_smiles(
    "[H][C]([H])([H])[C@@]([H])([C](=[O])[O-])[N+]([H])([H])[H]",
    hydrogens_are_explicit=True
)

assert zw_l_alanine.is_isomorphic_with(smiles_explicit_h)

smiles_explicit_h.visualize()
```

6.1.3 Mapped SMILES

By default, no guarantees are made about the indexing of atoms from a SMILES string. If the indexing is important, a mapped SMILES string may be used. In this case, Hydrogens must be explicit. Note that though mapped SMILES strings must start at index 1, Python lists start at index 0.

```
mapped_smiles = Molecule.from_mapped_smiles(
    "[H:10][C:2]([H:7])([H:8])[C@@:4]([H:9])([C:3](=[O:5])[O-:6])[N+:1]([H:11])([H:12])[H:13]
↪"
)

assert zw_l_alanine.is_isomorphic_with(mapped_smiles)

assert mapped_smiles.atoms[0].atomic_number == 7 # First index is the Nitrogen
assert all([a.atomic_number==1 for a in mapped_smiles.atoms[6:]]) # Final indices are all H

mapped_smiles.visualize()
```

6.1.4 SMILES without stereochemistry

The Toolkit won't accept an ambiguous SMILES. This SMILES could be L- or D- alanine; rather than guess, the Toolkit throws an error:

```
smiles_non_isomeric = Molecule.from_smiles(
    "CC([NH3+])C(=O)[O-]"
)
```

We can downgrade this error to a warning with the `allow_undefined_stereo` argument. This will not apply an improper dihedral term to the stereocenter and may lead to simulations with unphysical stereoisomerisation.

```
smiles_non_isomeric = Molecule.from_smiles(
    "CC([NH3+])C(=O)[O-]",
    allow_undefined_stereo=True
)

assert not zw_l_alanine.is_isomorphic_with(smiles_non_isomeric)

smiles_non_isomeric.visualize()
```

6.2 By hand

You can always construct a `Molecule` by building it up from individual atoms and bonds. Other methods are generally easier, but it's a useful fallback for when you need to write your own constructor for an unsupported source format.

The `Molecule()` constructor and the `add_atom()` and `add_bond()` methods are used to construct a `Molecule` by hand.

```
by_hand = Molecule()
by_hand.name = "Zwitterionic l-Alanine"

by_hand.add_atom(
    atomic_number = 8, # Atomic number 8 is Oxygen
    formal_charge = -1, # Formal negative charge
    is_aromatic = False, # Atom is not part of an aromatic system
    stereochemistry = None, # Optional argument; "R" or "S" stereochemistry
    name = "O-" # Optional argument; descriptive name for the atom
)
by_hand.add_atom(6,  0, False, name="C")
by_hand.add_atom(8,  0, False, name="O")
by_hand.add_atom(6,  0, False, stereochemistry="S", name="CA")
by_hand.add_atom(1,  0, False, name="CAH")
by_hand.add_atom(6,  0, False, name="CB")
by_hand.add_atom(1,  0, False, name="HB1")
by_hand.add_atom(1,  0, False, name="HB2")
by_hand.add_atom(1,  0, False, name="HB3")
by_hand.add_atom(7, +1, False, name="N+")
by_hand.add_atom(1,  0, False, name="HN1")
by_hand.add_atom(1,  0, False, name="HN2")
```

(continues on next page)

(continued from previous page)

```
by_hand.add_atom(1,  0, False, name="HN3")


by_hand.add_bond(
    atom1 = 0, # First (zero-indexed) atom specified above ("O-")
    atom2 = 1, # Second atom specified above ("C")
    bond_order = 1, # Single bond
    is_aromatic = False, # Bond is not aromatic
    stereochemistry = None, # Optional argument; "E" or "Z" stereochemistry
    fractional_bond_order = None # Optional argument; Wiberg (or similar) bond order
)
by_hand.add_bond( 1,  2, 2, False) # C = O
by_hand.add_bond( 1,  3, 1, False) # C - CA
by_hand.add_bond( 3,  4, 1, False) # CA - CAH
by_hand.add_bond( 3,  5, 1, False) # CA - CB
by_hand.add_bond( 5,  6, 1, False) # CB - HB1
by_hand.add_bond( 5,  7, 1, False) # CB - HB2
by_hand.add_bond( 5,  8, 1, False) # CB - HB3
by_hand.add_bond( 3,  9, 1, False) # CB - N+
by_hand.add_bond( 9, 10, 1, False) # N+ - HN1
by_hand.add_bond( 9, 11, 1, False) # N+ - HN2
by_hand.add_bond( 9, 12, 1, False) # N+ - HN3

assert zw_l_alanine.is_isomorphic_with(by_hand)

by_hand.visualize()
```

6.2.1 From a dictionary

Rather than build up the `Molecule` one method at a time, the `Molecule.from_dict()` method can construct a `Molecule` in one shot from a Python `dict` that describes the molecule in question. This allows `Molecule` objects to be written to and read from disk in any format that can be interpreted as a `dict`; this mechanism underlies the `from_bson()`, `from_json()`, `from_messagepack()`, `from_pickle()`, `from_toml()`, `from_xml()`, and `from_yaml()` methods.

This format can get very verbose, as it is intended for serialization, so this example uses hydrogen cyanide rather than alanine.

```
molecule_dict = {
    "name": "",
    "atoms": [
        {
            "atomic_number": 1,
            "formal_charge": 0,
            "is_aromatic": False,
            "stereochemistry": None,
            "name": "H",
        },
        {
            "atomic_number": 6,
            "formal_charge": 0,
```

(continues on next page)

(continued from previous page)

```
            "is_aromatic": False,
            "stereochemistry": None,
            "name": "C",
        },
        {
            "atomic_number": 7,
            "formal_charge": 0,
            "is_aromatic": False,
            "stereochemistry": None,
            "name": "N",
        },
    ],
    "virtual_sites": [],
    "bonds": [
        {
            "atom1": 0,
            "atom2": 1,
            "bond_order": 1,
            "is_aromatic": False,
            "stereochemistry": None,
            "fractional_bond_order": None,
        },
        {
            "atom1": 1,
            "atom2": 2,
            "bond_order": 3,
            "is_aromatic": False,
            "stereochemistry": None,
            "fractional_bond_order": None,
        },
    ],
    "properties": {},
    "conformers": None,
    "partial_charges": None,
    "partial_charges_unit": None,
}

from_dictionary = Molecule.from_dict(molecule_dict)

from_dictionary.visualize()
```

6.3 From a file

We can construct a `Molecule` from a file or file-like object with the `from_file()` method. We're a bit constrained in what file formats we can accept, because they need to provide all the information needed to construct the molecular graph; not just coordinates, but also elements, formal charges, bond orders, and stereochemistry.

6.3.1 From SDF file

We generally recommend the SDF format. The SDF file used here can be found on GitHub

```
sdf_path = Molecule.from_file("zw_l_alanine.sdf")
assert zw_l_alanine.is_isomorphic_with(sdf_path)
sdf_path.visualize()
```

6.3.2 From SDF file object

`from_file()` can also take a file object, rather than a path. Note that the object must be in binary mode!

```
with open("zw_l_alanine.sdf", mode="rb") as file:
    sdf_object = Molecule.from_file(file, file_format="SDF")

assert zw_l_alanine.is_isomorphic_with(sdf_object)
sdf_object.visualize()
```

6.3.3 From PDB file

Using PDB files is not recommended, even if they have CONECT records, as they do not provide stereoisomeric information or bond orders. The RDKit backend assumes that bond orders are 1, so the toolkit refuses to use it:

```
from openff.toolkit.utils.toolkits import RDKitToolkitWrapper

pdb = Molecule.from_file("zw_l_alanine.pdb", "pdb", toolkit_registry=RDKitToolkitWrapper())
```

OpenEye can infer bond orders and stereochemistry from the structure. This is not recommended, as it can make mistakes that may be difficult to catch. Note also that this requires a license for OpenEye, as this is proprietary software.

If we instead provide a SMILES code, a PDB file can be used to populate the `Molecule` object's `conformers` attribute and provide atom ordering, as well as check that the SMILES code matches the PDB file. This method is the recommended way to create a `Molecule` from a PDB file. The PDB file used here can be found on GitHub

---

Important: Note that the Toolkit doesn't guarantee that the coordinates in the PDB are correctly assigned to atoms. It makes an effort, but you should check that the results are reasonable.

```
pdb_with_smiles = Molecule.from_pdb_and_smiles(
    "zw_l_alanine.pdb",
    "C[C@H]([NH3+])C(=O)[O-]"
)

assert zw_l_alanine.is_isomorphic_with(pdb_with_smiles)

pdb_with_smiles.visualize()
```

6.4 Other string identification formats

The OpenFF Toolkit supports a few text based molecular identity formats other than SMILES (*see above*)

6.4.1 From InChI

The `Molecule.from_inchi()` method constructs a `Molecule` from an IUPAC InChI string. Note that InChI cannot distinguish the zwitterionic form of alanine from the neutral form (see section 13.2 of the InChI Technical FAQ), so the toolkit defaults to the neutral form.

Warning: The OpenFF Toolkit makes no guarantees about the atomic ordering produced by the `from_inchi` method. InChI is not intended to be an interchange format.

```
inchi = Molecule.from_inchi("InChI=1S/C3H7NO2/c1-2(4)3(5)6/h2H,4H2,1H3,(H,5,6)/t2-/m0/s1")

inchi.visualize()
```

6.4.2 From IUPAC name

The `Molecule.from_iupac()` method constructs a `Molecule` from an IUPAC name.

Important: This code requires the OpenEye toolkit.

```
iupac = Molecule.from_iupac("(2S)-2-azaniumylpropanoate")

assert zw_l_alanine.is_isomorphic_with(iupac)

iupac.visualize()
```

6.5 Remapping an existing `Molecule`

Most `Molecule` creation methods don't specify the ordering of atoms in the new `Molecule`. The `Molecule.remap()` method allows a new ordering to be applied to an existing `Molecule`.

See also *Mapped SMILES*.

Warning: The `Molecule.remap()` method is experimental and subject to change.

```
# Note that this mapping is off-by-one from the mapping taken
# by the remap method, as Python indexing is 0-based but SMILES
# is 1-based
print("Before remapping:", zw_l_alanine.to_smiles(mapped=True))

# Flip the positions of the oxygen atoms
remapped = zw_l_alanine.remap({0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 6, 6: 5, 7: 7, 8: 8, 9: 9,␣
↪10: 10, 11: 11, 12: 12})

print("After remapping: ", remapped.to_smiles(mapped=True))

# Doesn't affect the identity of the molecule
assert zw_l_alanine.is_isomorphic_with(remapped)
remapped.visualize()
```

6.6 Via `Topology` objects

The `Topology` class represents a biomolecular system; it is analogous to the similarly named objects in GROMACS, MDTraj or OpenMM. Notably, it does not include co-ordinates and may represent multiple copies of a particular molecular species or even more complex mixtures of molecules. `Topology` objects are usually built up one species at a time from `Molecule` objects.

The `Molecule.from_topology()` method constructs a `Molecule` from a `Topology`. This is usually going backwards, but the method does allow construction of `Molecule` objects from a few sources that represent molecular mixtures, like the aforementioned `Topology` or `System`.

Constructor methods that are available for `Topology` but not `Molecule` generally require a `Molecule` to be provided via the `unique_molecules` keyword argument. The provided `Molecule` is used to provide the identity of the molecule, including aromaticity, bond orders, formal charges, and so forth. These methods therefore don't provide a route to the graph of the molecule, but can be useful for reordering atoms to match another software package.

6.6.1 From an OpenMM Topology

The Topology.from_openmm() method constructs an OpenFF Topology from an OpenMM Topology. The method requires that all the unique molecules in the Topology are provided as OpenFF Molecule objects, as the structure of an OpenMM Topology doesn't include the concept of a molecule. When using this method to create a Molecule, this limitation means that the method really only offers a pathway to reorder the atoms of a Molecule to match that of the OpenMM Topology.

```
from simtk.openmm.app.pdbfile import PDBFile

openmm_topology = PDBFile('zw_l_alanine.pdb').getTopology()
openff_topology = Topology.from_openmm(openmm_topology, unique_molecules=[zw_l_alanine])

from_openmm_topology = Molecule.from_topology(openff_topology)

print(zw_l_alanine.to_smiles(mapped=True))
print(from_openmm_topology.to_smiles(mapped=True))

from_openmm_topology.visualize()
```

6.6.2 From an MDTraj Topology

The Topology.from_mdtraj() method constructs an OpenFF Topology from an MDTraj Topology. The method requires that all the unique molecules in the Topology are provided as OpenFF Molecule objects to ensure that the graph of the molecule is correct. When using this method to create a Molecule, this limitation means that the method really only offers a pathway to reorder the atoms of a Molecule to match that of the MDTraj Topology.

```
from mdtraj import load_pdb

mdtraj_topology = load_pdb('zw_l_alanine.pdb').topology
openff_topology = Topology.from_openmm(openmm_topology, unique_molecules=[zw_l_alanine])

from_mdtraj_topology = Molecule.from_topology(openff_topology)

print(zw_l_alanine.to_smiles(mapped=True))
print(from_mdtraj_topology.to_smiles(mapped=True))

from_mdtraj_topology.visualize()
```

6.7 From Toolkit objects

The OpenFF Toolkit calls out to other software to perform low-level tasks like reading SMILES or files. These external software packages are called toolkits, and presently include RDKit and the OpenEye Toolkit. OpenFF Molecule objects can be created from the equivalent objects in these toolkits.

6.7.1 From RDKit `Mol`

The `Molecule.from_rdkit()` method converts an `rdkit.Chem.rdchem.Mol` object to an OpenFF `Molecule`.

```
from rdkit import Chem
rdmol = Chem.MolFromSmiles("C[C@H]([NH3+])C([O-])=O")

print("rdmol is of type", type(rdmol))

from_rdmol = Molecule.from_rdkit(rdmol)

assert zw_l_alanine.is_isomorphic_with(from_rdmol)
from_rdmol.visualize()
```

6.7.2 From OpenEye `OEMol`

The `Molecule.from_openeye()` method converts an object that inherits from `openeye.oechem.OEMolBase` to an OpenFF `Molecule`.

```
from openeye import oechem

oemol = oechem.OEGraphMol()
oechem.OESmilesToMol(oemol, "C[C@H]([NH3+])C([O-])=O")

assert isinstance(oemol, oechem.OEMolBase)

from_oemol = Molecule.from_openeye(oemol)

assert zw_l_alanine.is_isomorphic_with(from_oemol)
from_oemol.visualize()
```

6.8 From QCArchive

QCArchive is a repository of quantum chemical calculations on small molecules. The `Molecule.from_qcschema()` method creates a `Molecule` from a record from the archive. Because the identity of a molecule can change of the course of a QC calculation, the Toolkit accepts records only if they contain a hydrogen-mapped SMILES code.

Note: These examples use molecules other than l-Alanine because of their availability in QCArchive

6.8.1 From a QCArchive molecule record

The `Molecule.from_qcschema()` method can take a molecule record queried from the QCArchive and create a `Molecule` from it.

```
from qcportal import FractalClient

client = FractalClient()
query = client.query_molecules(molecular_formula="C16H20N3O5")

from_qcarchive = Molecule.from_qcschema(query[0])

from_qcarchive.visualize()
```

6.8.2 From a QCArchive optimisation record

`Molecule.from_qcschema()` can also take an optimisation record and create the corresponding `Molecule`.

```
optimization_dataset = client.get_collection(
    "OptimizationDataset",
    "SMIRNOFF Coverage Set 1"
)
dimethoxymethanol_optimization = optimization_dataset.get_entry('coc(o)oc-0')

from_optimisation = Molecule.from_qcschema(dimethoxymethanol_optimization)

from_optimisation.visualize()
```

CHAPTER

SEVEN

THE SMIRKS NATIVE OPEN FORCE FIELD (SMIRNOFF) SPECIFICATION

SMIRNOFF is a specification for encoding molecular mechanics force fields from the Open Force Field Initiative based on direct chemical perception using the broadly-supported SMARTS language, utilizing atom tagging extensions from SMIRKS.

7.1 Authors and acknowledgments

The SMIRNOFF specification was designed by the Open Force Field Initiative.

Primary contributors include:

- Caitlin C. Bannan (University of California, Irvine) <bannanc@uci.edu>
- Christopher I. Bayly (OpenEye Software) <bayly@eyesopen.com>
- John D. Chodera (Memorial Sloan Kettering Cancer Center) <john.chodera@choderalab.org>
- David L. Mobley (University of California, Irvine) <dmobley@uci.edu>

SMIRNOFF and its reference implementation in the Open Force Field Toolkit was heavily inspired by the ForceField class from the OpenMM molecular simulation package, and its associated XML format, developed by Peter K. Eastman (Stanford University).

7.2 Representations and encodings

A force field in the SMIRNOFF format can be encoded in multiple representations. Currently, only an XML representation is supported by the reference implementation of the OpenFF toolkit.

7.2.1 XML representation

A SMIRNOFF force field can be described in an XML representation, which provides a human- and machine-readable form for encoding the parameter set and its typing rules. This document focuses on describing the XML representation of the force field.

- By convention, XML-encoded SMIRNOFF force fields use an `.offxml` extension if written to a file to prevent confusion with other file formats.
- In XML, numeric quantities appear as strings, like `"1"` or `"2.3"`.
- Integers should always be written without a decimal point, such as `"1"`, `"9"`.
- Non-integral numbers, such as parameter values, should be written with a decimal point, such as `"1.23"`, `"2."`.

- In XML, certain special characters that occur in valid SMARTS/SMIRKS patterns (such as ampersand symbols `&`) must be specially encoded. See this list of XML and HTML character entity references for more details.

7.2.2 Future representations: JSON, MessagePack, YAML, and TOML

We are considering supporting JSON, MessagePack, YAML, and TOML representations as well.

7.3 Reference implementation

A reference implementation of the SMIRNOFF XML specification is provided in the OpenFF toolkit.

7.4 Support for molecular simulation packages

The reference implementation currently generates parameterized molecular mechanics systems for the GPU-accelerated OpenMM molecular simulation toolkit. Parameterized systems can subsequently be converted for use in other popular molecular dynamics simulation packages (including AMBER, CHARMM, NAMD, Desmond, and LAMMPS) via ParmEd and InterMol. See the example on using SMIRNOFF in AMBER or GROMACS for more details.

7.5 Basic structure

A reference implementation of a SMIRNOFF force field parser that can process XML representations (denoted by `.offxml` file extensions) can be found in the `ForceField` class of the `openff.toolkit.typing.engines.smirnoff` module.

Below, we describe the main structure of such an XML representation.

7.5.1 The enclosing <SMIRNOFF> tag

A SMIRNOFF force field XML specification always is enclosed in a `<SMIRNOFF>` tag, with certain required attributes provided. The required and permitted attributes defined in the `<SMIRNOFF>` are recorded in the version attribute, which describes the top-level attributes that are expected or permitted to be defined.

```
<SMIRNOFF version="0.3" aromaticity_model="OEAroModel_MDL">
...
</SMIRNOFF>
```

Versioning

The SMIRNOFF force field format supports versioning via the `version` attribute to the root `<SMIRNOFF>` tag, e.g.:

```
<SMIRNOFF version="0.3" aromaticity_model="OEAroModel_MDL">
...
</SMIRNOFF>
```

The version format is `x.y`, where `x` denotes the major version and `y` denotes the minor version. SMIRNOFF versions are guaranteed to be backward-compatible within the *same major version number series*, but it is possible major version increments will break backwards-compatibility.

SMIRNOFF tag version	Required attributes	Optional attributes
0.1	`aromaticity_model`	`Date`, `Author`
0.2	`aromaticity_model`	`Date`, `Author`
0.3	`aromaticity_model`	`Date`, `Author`

The SMIRNOFF tag versions describe the required and allowed force field-wide settings. The list of keywords is as follows:

Aromaticity model

The `aromaticity_model` specifies the aromaticity model used for chemical perception (here, `OEAroModel_MDL`).

Currently, the only supported model is `OEAroModel_MDL`, which is implemented in both the RDKit and the OpenEye Toolkit.

Note: Add link to complete open specification of `OEAroModel_MDL` aromaticity model.

7.5.2 Metadata

Typically, date and author information is included:

```
<Date>2016-05-25</Date>
<Author>J. D. Chodera (MSKCC) charge increment tests</Author>
```

The `<Date>` tag should conform to ISO 8601 date formatting guidelines, such as `2018-07-14` or `2018-07-14T08:50:48+00:00` (UTC time).

7.5.3 Parameter generators

Within the <SMIRNOFF> tag, top-level tags encode parameters for a force field based on a SMARTS/SMIRKS-based specification describing the chemical environment the parameters are to be applied to. The file has tags corresponding to OpenMM force terms (Bonds, Angles, ProperTorsions, etc., as discussed in more detail below); these specify functional form and other information for individual force terms.

```
<Angles version="0.3" potential="harmonic">
   ...
</Angles>
```

which introduces the following Angle child elements which will use a harmonic potential.

7.5.4 Specifying parameters

Under each of these force terms, there are tags for individual parameter lines such as these:

```
<Angles version="0.3" potential="harmonic">
   <Angle smirks="[a,A:1]-[#6X4:2]-[a,A:3]" angle="109.50*degree" k="100.0*kilocalorie_per_
↪mole/radian**2"/>
   <Angle smirks="[#1:1]-[#6X4:2]-[#1:3]" angle="109.50*degree" k="70.0*kilocalorie_per_mole/
↪radian**2"/>
</Angles>
```

The first of these specifies the smirks attribute as [a,A:1]-[#6X4:2]-[a,A:3], specifying a SMIRKS pattern that matches three connected atoms specifying an angle. This particular SMIRKS pattern matches a tetravalent carbon at the center with single bonds to two atoms of any type. This pattern is essentially a SMARTS string with numerical atom tags commonly used in SMIRKS to identify atoms in chemically unique environments—these can be thought of as tagged regular expressions for identifying chemical environments, and atoms within those environments. Here, [a,A] denotes any atom—either aromatic (a) or aliphatic (A), while [#6X4] denotes a carbon by element number (#6) that with four substituents (X4). The symbol - joining these groups denotes a single bond. The strings :1, :2, and :2 label these atoms as indices 1, 2, and 3, with 2 being the central atom. Equilibrium angles are provided as the angle attribute, along with force constants as the k attribute (with corresponding units included in the expression).

Note: The reference implementation of the SMIRNOFF specification implemented in the Open Force Field Toolkit will, by default, raise an exception if an unexpected attribute is encountered. The toolkit can be configured to accept non-spec keywords, but these are considered "cosmetic" and will not be evaluated. For example, providing an <Angle> tag that also specifies a second force constant k2 will result in an exception, unless the user specifies that "cosmetic" attributes should be accepted by the parser.

7.5.5 SMIRNOFF parameter specification is hierarchical

Parameters that appear later in a SMIRNOFF specification override those which come earlier if they match the same pattern. This can be seen in the example above, where the first line provides a generic angle parameter for any tetravalent carbon (single bond) angle, and the second line overrides this for the specific case of a hydrogen-(tetravalent carbon)-hydrogen angle. This hierarchical structure means that a typical parameter file will tend to have generic parameters early in the section for each force type, with more specialized parameters assigned later.

7.5.6 Multiple SMIRNOFF representations can be processed in sequence

Multiple SMIRNOFF data sources (e.g. multiple OFFXML files) can be loaded by the OpenFF `ForceField` in sequence. If these files each contain unique top-level tags (such as `<Bonds>`, `<Angles>`, etc.), the resulting force field will be independent of the order in which the files are loaded. If, however, the same tag occurs in multiple files, the contents of the tags are merged, with the tags read later taking precedence over the parameters read earlier, provided the top-level tags have compatible attributes. The resulting force field will therefore depend on the order in which parameters are read.

This behavior is intended for limited use in appending very specific parameters, such as parameters specifying solvent models, to override standard parameters.

7.6 Units

To minimize the potential for unit conversion errors, SMIRNOFF force fields explicitly specify units in a form readable to both humans and computers for all unit-bearing quantities. Allowed values for units are given in simtk.unit (though in the future this may change to the more widely-used Python pint library). For example, for the `angle` (equilibrium angle) and `k` (force constant) parameters in the `<Angle>` example block above, both attributes are specified as a mathematical expression

```
   <Angle smirks="[#1:1]-[#6X4:2]-[#1:3]" angle="109.50*degree" k="70.0*kilocalorie_per_mole/
↪radian**2"/>
```

For more information, see the standard OpenMM unit system.

7.7 SMIRNOFF independently applies parameters to each class of potential energy terms

The SMIRNOFF uses direct chemical perception to assign parameters for potential energy terms independently for each term. Rather than first applying atom typing rules and then looking up combinations of the resulting atom types for each force term, the rules for directly applying parameters to atoms is compartmentalized in separate sections. The file consists of multiple top-level tags defining individual components of the potential energy (in addition to charge models or modifiers), with each section specifying the typing rules used to assign parameters for that potential term:

```
<Bonds version="0.3" potential="harmonic">
   <Bond smirks="[#6X4:1]-[#6X4:2]" length="1.526*angstrom" k="620.0*kilocalories_per_mole/
↪angstrom**2"/>
   <Bond smirks="[#6X4:1]-[#1:2]" length="1.090*angstrom" k="680.0*kilocalories_per_mole/
↪angstrom**2"/>
   ...
</Bonds>

<Angles version="0.3" potential="harmonic">
   <Angle smirks="[a,A:1]-[#6X4:2]-[a,A:3]" angle="109.50*degree" k="100.0*kilocalories_per_
↪mole/radian**2"/>
   <Angle smirks="[#1:1]-[#6X4:2]-[#1:3]" angle="109.50*degree" k="70.0*kilocalories_per_
↪mole/radian**2"/>
   ...
</Angles>
```

Each top-level tag specifying a class of potential energy terms has an attribute `potential` for specifying the functional form for the interaction. Common defaults are defined, but the goal is to eventually allow these to be overridden by alternative choices or even algebraic expressions in the future, once more molecular simulation packages support general expressions. We distinguish between functional forms available in all common molecular simulation packages (specified by keywords) and support for general functional forms available in a few packages (especially OpenMM, which supports a flexible set of custom forces defined by algebraic expressions) with an **EXPERIMENTAL** label.

Many of the specific forces are implemented as discussed in the OpenMM Documentation; see especially Section 19 on Standard Forces for mathematical descriptions of these functional forms. Some top-level tags provide attributes that modify the functional form used to be consistent with packages such as AMBER or CHARMM.

7.8 Partial charge and electrostatics models

SMIRNOFF supports several approaches to specifying electrostatic models. Currently, only classical fixed point charge models are supported, but future extensions to the specification will support point multipoles, point polarizable dipoles, Drude oscillators, charge equilibration methods, and so on.

7.8.1 <LibraryCharges>: Library charges for polymeric residues and special solvent models

A mechanism is provided for specifying library charges that can be applied to molecules or residues that match provided templates. Library charges are applied first, and atoms for which library charges are applied will be excluded from alternative charging schemes listed below.

For example, to assign partial charges for a non-terminal ALA residue from the AMBER ff14SB parameter set:

```
<LibraryCharges version="0.3">
   <!-- match a non-terminal alanine residue with AMBER ff14SB partial charges -->
   <LibraryCharge name="ALA" smirks="[NX3:1]([#1:2])([#6])[#6H1:3]([#1:4])([#6:5]([#1:6])([
↪#1:7])[#1:8])[#6:9](=[#8:10])[#7]" charge1="-0.4157*elementary_charge" charge2="0.
↪2719*elementary_charge" charge3="0.0337*elementary_charge" charge4="0.0823*elementary_
↪charge" charge5="-0.1825*elementary_charge" charge6="0.0603*elementary_charge" charge7="0.
↪0603*elementary_charge" charge8="0.0603*elementary_charge" charge9="0.5973*elementary_
↪charge" charge10="-0.5679*elementary_charge"/>
   ...
</LibraryCharges>
```

In this case, a SMIRKS string defining the residue tags each atom that should receive a partial charge, with the charges specified by attributes `charge1`, `charge2`, etc. The `name` attribute is optional. Note that, for a given template, chemically equivalent atoms should be assigned the same charge to avoid undefined behavior. If the template matches multiple non-overlapping sets of atoms, all such matches will be assigned the provided charges. If multiple templates match the same set of atoms, the last template specified will be used.

Solvent models or excipients can also have partial charges specified via the `<LibraryCharges>` tag. For example, to ensure water molecules are assigned partial charges for TIP3P water, we can specify a library charge entry:

```
<LibraryCharges version="0.3">
   <!-- TIP3P water oxygen with charge override -->
```

(continues on next page)

(continued from previous page)

```
  <LibraryCharge name="TIP3P" smirks="[#1:1]-[#8X2H2+0:2]-[#1:3]" charge1="0.417*elementary_
↪charge" charge2="-0.834*elementary_charge" charge3="0.417*elementary_charge"/>
</LibraryCharges>
```

LibraryCharges section tag version	Tag attributes and default values	Required parameter attributes	Optional parameter attributes
0.3		`smirks`, `charge` (indexed)	`name`, `id`, `parent_id`

7.8.2 <ChargeIncrementModel>: Small molecule and fragment charges

In keeping with the AMBER force field philosophy, especially as implemented in small molecule force fields such as GAFF, GAFF2, and parm@Frosst, partial charges for small molecules are usually assigned using a quantum chemical method (usually a semiempirical method such as AM1) and a partial charge determination scheme (such as CM2 or RESP), then subsequently corrected via charge increment rules, as in the highly successful AM1-BCC approach.

Here is an example:

```
<ChargeIncrementModel version="0.4" number_of_conformers="1" partial_charge_method="AM1-
↪Mulliken">
  <!-- A fractional charge can be moved along a single bond -->
  <ChargeIncrement smirks="[#6X4:1]-[#6X3a:2]" charge_increment1="-0.0073*elementary_charge"␣
↪charge_increment2="0.0073*elementary_charge"/>
  <ChargeIncrement smirks="[#6X4:1]-[#6X3a:2]-[#7]" charge_increment1="0.0943*elementary_
↪charge" charge_increment2="-0.0943*elementary_charge"/>
  <!--- Alternatively, fractional charges can be redistributed among any number of bonded␣
↪atoms -->
  <ChargeIncrement smirks="[N:1]([H:2])([H:3])" charge_increment1="0.02*elementary_charge"␣
↪charge_increment2="-0.01*elementary_charge" charge_increment3="-0.01*elementary_charge"/>
  <!-- As of version 0.4 of the ChargeIncrementModel tag, it is possible to define one less␣
↪charge_increment attribute than there are tagged atoms -->
  <!-- The final, undefined charge_increment will be calculated as to make the sum of the␣
↪charge_increments equal 0 -->
  <ChargeIncrement smirks="[#6X4:1]-[#8:2]" charge_increment1="-0.0718*elementary_charge"/>
  <ChargeIncrement smirks="[N]-[C:1]-[C:2]-[Cl:3]" charge_increment1="-0.123" charge_
↪increment2="0.456" />
</ChargeIncrementModel>
```

The sum of formal charges for the molecule or fragment will be used to determine the total charge the molecule or fragment will possess.

`<ChargeIncrementModel>` provides several optional attributes to control its behavior:

- The `number_of_conformers` attribute (default: `"1"`) is used to specify how many conformers will be generated for the molecule (or capped fragment) prior to charging.
- The `partial_charge_method` attribute (default: `"AM1-Mulliken"`) is used to specify how uncorrected partial charges are to be generated. Later additions will add restrained electrostatic potential fitting (RESP) capabilities.

The `<ChargeIncrement>` tags specify how the quantum chemical derived charges are to be corrected to produce the final charges. The `charge_increment#` attributes specify how much the charge on the associated

tagged atom index (replacing #) should be modified.

Starting in the 0.4 version of this section, a `ChargeIncrement` may be specified with one less `charge_increment` value than it has tagged atoms. The missing `charge_increment` value must be that of the highest tagged atom index. This missing `charge_increment` will be calculated to offset the sum of the other `charge_increments` in the same `ChargeIncrement` parameter to achieve a net value of 0. This allows `ChargeIncrement` parameters to be defined similar to bond charge corrections.

Note that atoms for which library charges have already been applied are excluded from charging via `<ChargeIncrementModel>`.

Future additions will provide options for intelligently fragmenting large molecules and biopolymers, as well as a `capping` attribute to specify how fragments with dangling bonds are to be capped to allow these groups to be charged.

ChargeIncrementModel section tag version	Tag attributes and default values	Required parameter attributes	Optional parameter attributes
0.3	`number_of_conformers="1"`, `partial_charge_method='AM1-Mulliken'`	`smirks`, `charge_increment` (indexed, must be equal to number of tagged atoms in `smirks`)	`name`, `id`, `parent_id`
0.4	`number_of_conformers="1"`, `partial_charge_method='AM1-Mulliken'`	`smirks`, `charge_increment` (indexed, must be equal to- or one less than- number of tagged atoms in `smirks`)	`name`, `id`, `parent_id`

7.8.3 `<ToolkitAM1BCC>`: Temporary support for toolkit-based AM1-BCC partial charges

Warning: This tag is not permanent and may be phased out in future versions of the spec.

This tag calculates partial charges using the default settings of the highest-priority cheminformatics toolkit that can perform AM1-BCC charge assignment. Currently, if the OpenEye toolkit is licensed and available, this will use QuacPac configured to generate charges using AM1-BCC ELF10 for each unique molecule in the topology. Otherwise RDKit will be used for initial conformer generation and the AmberTools antechamber/sqm software will be used for charge calculation.

If this tag is specified for a force field, conformer generation will be performed regardless of whether conformations of the input molecule were provided. If RDKit/AmberTools are used as the toolkit backend for this calculation, only the first conformer is used for AM1-BCC calculation.

The charges generated by this tag may differ depending on which toolkits are available.

Note that atoms for which prespecified or library charges have already been applied are excluded from charging via `<ToolkitAM1BCC>`.

7.8.4 Prespecified charges (reference implementation only)

In our reference implementation of SMIRNOFF in the Open Force Field Toolkit, we also provide a method for specifying user-defined partial charges during system creation. This functionality is accessed by using the `charge_from_molecules` optional argument during system creation, such as in `ForceField.create_openmm_system(topology, charge_from_molecules=molecule_list)`. When this optional keyword is provided, all matching molecules will have their charges set by the entries in `molecule_list`. This method is provided solely for convenience in developing and exploring alternative charging schemes; actual force field releases for distribution will use one of the other mechanisms specified above.

7.9 Parameter sections

A SMIRNOFF force field consists of one or more force field term definition sections. For the most part, these sections independently define how a specific component of the potential energy function for a molecular system is supposed to be computed (such as bond stretch energies, or Lennard-Jones interactions), as well as how parameters are to be assigned for this particular term. Each parameter section contains a `version`, which encodes the behavior of the section, as well as the required and optional attributes the top-level tag and SMIRKS-based parameters. This decoupling of how parameters are assigned for each term provides a great deal of flexibility in composing new force fields while allowing a minimal number of parameters to be used to achieve accurate modeling of intramolecular forces.

Below, we describe the specification for each force field term definition using the XML representation of a SMIRNOFF force field.

As an example of a complete SMIRNOFF force field specification, see a recent force field in the "Parsley" line (openff-1.2.0.offxml).

Note: Not all parameter sections *must* be specified in a SMIRNOFF force field. A wide variety of force field terms are provided in the specification, but a particular force field only needs to define a subset of those terms.

7.9.1 <vdW>

van der Waals force parameters, which include repulsive forces arising from Pauli exclusion and attractive forces arising from dispersion, are specified via the `<vdW>` tag with sub-tags for individual `Atom` entries, such as:

```
<vdW version="0.3" potential="Lennard-Jones-12-6" combining_rules="Lorentz-Berthelot"␣
↪scale12="0.0" scale13="0.0" scale14="0.5" scale15="1.0" switch_width="8.0*angstrom" cutoff=
↪"9.0*angstrom" long_range_dispersion="isotropic">
   <Atom smirks="[#1:1]" sigma="1.4870*angstrom" epsilon="0.0157*kilocalories_per_mole"/>
   <Atom smirks="[#1:1]-[#6]" sigma="1.4870*angstrom" epsilon="0.0157*kilocalories_per_mole"/
↪>
   ...
</vdW>
```

For standard Lennard-Jones 12-6 potentials (specified via `potential="Lennard-Jones-12-6"`), the `epsilon` parameter denotes the well depth, while the size property can be specified either via providing the `sigma` attribute, such as `sigma="1.3*angstrom"`, or via the `r_0/2` (`rmin/2`) values used in AMBER force fields (here denoted `rmin_half` as in the example above). The two are related by `r0 = 2^(1/6)*sigma` and conversion is done internally in `ForceField` into the `sigma` values used in OpenMM.

Attributes in the <vdW> tag specify the scaling terms applied to the energies of 1-2 (scale12, default: 0), 1-3 (scale13, default: 0), 1-4 (scale14, default: 0.5), and 1-5 (scale15, default: 1.0) interactions, as well as the distance at which a switching function is applied (switch_width, default: "1.0*angstrom"), the cutoff (cutoff, default: "9.0*angstroms"), and long-range dispersion treatment scheme (long_range_dispersion, default: "isotropic").

The potential attribute (default: "none") specifies the potential energy function to use. Currently, only potential="Lennard-Jones-12-6" is supported:

```
U(r) = 4*epsilon*((sigma/r)^12 - (sigma/r)^6)
```

The combining_rules attribute (default: "none") currently only supports "Lorentz-Berthelot", which specifies the geometric mean of epsilon and arithmetic mean of sigma. Support for other Lennard-Jones mixing schemes will be added later: Waldman-Hagler, Fender-Halsey, Kong, Tang-Toennies, Pena, Hudson-McCoubrey, Sikora.

Later revisions will add support for additional potential types (e.g., Buckingham-exp-6), as well as the ability to support arbitrary algebraic functional forms using a scheme such as

```
<vdW version="0.3" potential="4*epsilon*((sigma/r)^12-(sigma/r)^6)" scale12="0.0" scale13="0.
↪0" scale14="0.5" scale15="1" switch_width="8.0*angstrom" cutoff="9.0*angstrom" long_range_
↪dispersion="isotropic">
   <CombiningRules>
      <CombiningRule parameter="sigma" function="(sigma1+sigma2)/2"/>
      <CombiningRule parameter="epsilon" function="sqrt(epsilon1*epsilon2)"/>
   </CombiningRules>
   <Atom smirks="[#1:1]" sigma="1.4870*angstrom" epsilon="0.0157*kilocalories_per_mole"/>
   <Atom smirks="[#1:1]-[#6]" sigma="1.4870*angstrom" epsilon="0.0157*kilocalories_per_mole"/
↪>
   ...
</vdW>
```

If the <CombiningRules> tag is provided, it overrides the combining_rules attribute.

Later revisions will also provide support for special interactions using the <AtomPair> tag:

```
<vdW version="0.3" potential="Lennard-Jones-12-6" combining_rules="Lorentz-Berthelot"␣
↪scale12="0.0" scale13="0.0" scale14="0.5" scale15="1">
   <AtomPair smirks1="[#1:1]" smirks2="[#6:2]" sigma="1.4870*angstrom" epsilon="0.
↪0157*kilocalories_per_mole"/>
   ...
</vdW>
```

vdW section tag version	Tag attributes and default values	Required parameter attributes	Optional parameter attributes
0.3	potential="Lennard-Jones-12-6", combining_rules="Lorentz-Berthelot", scale12="0", scale13="0", scale14="0.5", scale15="1.0", cutoff="9.0*angstrom", switch_width="1.0*angstrom", method="cutoff"	smirks, epsilon, (sigma OR rmin_half)	id, parent_id

7.9.2 <Electrostatics>

Electrostatic interactions are specified via the <Electrostatics> tag.

```
<Electrostatics version="0.3" method="PME" scale12="0.0" scale13="0.0" scale14="0.833333"␣
↪scale15="1.0"/>
```

The method attribute specifies the manner in which electrostatic interactions are to be computed:

- PME - particle mesh Ewald should be used (DEFAULT); can only apply to periodic systems
- reaction-field - reaction-field electrostatics should be used; can only apply to periodic systems
- Coulomb - direct Coulomb interactions (with no reaction-field attenuation) should be used

The interaction scaling parameters applied to atoms connected by a few bonds are

- scale12 (default: 0) specifies the scaling applied to 1-2 bonds
- scale13 (default: 0) specifies the scaling applied to 1-3 bonds
- scale14 (default: 0.833333) specifies the scaling applied to 1-4 bonds
- scale15 (default: 1.0) specifies the scaling applied to 1-5 bonds

Currently, no child tags are used because the charge model is specified via different means (currently library charges or BCCs).

For methods where the cutoff is not simply an implementation detail but determines the potential energy of the system (reaction-field and Coulomb), the cutoff distance must also be specified, and a switch_width if a switching function is to be used.

Electrostatics section tag version	Tag attributes and default values	Required parameter attributes	Optional parameter attributes
0.3	scale12="0", scale13="0", scale14="0.833333", scale15="1.0", cutoff="9.0*angstrom", switch_width="0*angstrom", method="PME"	N/A	N/A

7.9.3 <Bonds>

Bond parameters are specified via a <Bonds>...</Bonds> block, with individual <Bond> tags containing attributes specifying the equilibrium bond length (length) and force constant (k) values for specific bonds. For example:

```
<Bonds version="0.3" potential="harmonic">
   <Bond smirks="[#6X4:1]-[#6X4:2]" length="1.526*angstrom" k="620.0*kilocalories_per_mole/
↪angstrom**2"/>
   <Bond smirks="[#6X4:1]-[#1:2]" length="1.090*angstrom" k="680.0*kilocalories_per_mole/
↪angstrom**2"/>
   ...
</Bonds>
```

Currently, only potential="harmonic" is supported, where we utilize the standard harmonic functional form:

```
U(r) = (k/2)*(r-length)^2
```

Later revisions will add support for additional potential types and the ability to support arbitrary algebraic functional forms. If the `potential` attribute is omitted, it defaults to `harmonic`.

Note that AMBER and CHARMM define a modified functional form, such that `U(r) = k*(r-length)^2`, so that force constants would need to be multiplied by two in order to be used in the SMIRNOFF format.

Constrained bonds are handled by a separate `<Constraints>` tag, which can either specify constraint distances or draw them from equilibrium distances specified in `<Bonds>`.

Fractional bond orders

Fractional bond orders can be used to allow interpolation of bond parameters. For example, these parameters:

```
<Bonds version="0.3" potential="harmonic">
    <Bond smirks="[#6X3:1]-[#6X3:2]" k="820.0*kilocalories_per_mole/angstrom**2" length="1.
↪45*angstrom"/>
    <Bond smirks="[#6X3:1]:[#6X3:2]" k="938.0*kilocalories_per_mole/angstrom**2" length="1.
↪40*angstrom"/>
    <Bond smirks="[#6X3:1]=[#6X3:2]" k="1098.0*kilocalories_per_mole/angstrom**2" length="1.
↪35*angstrom"/>
    ...
```

can be replaced by a single parameter line by first invoking the `fractional_bondorder_method` attribute to specify a method for computing the fractional bond order and `fractional_bondorder_interpolation` for specifying the procedure for interpolating parameters between specified integral bond orders:

```
<Bonds version="0.3" potential="harmonic" fractional_bondorder_method="AM1-Wiberg"␣
↪fractional_bondorder_interpolation="linear">
    <Bond smirks="[#6X3:1]!#[#6X3:2]" k_bondorder1="820.0*kilocalories_per_mole/angstrom**2"␣
↪k_bondorder2="1098*kilocalories_per_mole/angstrom**2" length_bondorder1="1.45*angstrom"␣
↪length_bondorder2="1.35*angstrom"/>
    ...
```

This allows specification of force constants and lengths for bond orders 1 and 2, and then interpolation between those based on the partial bond order.

- `fractional_bondorder_method` defaults to `AM1-Wiberg`.
- `fractional_bondorder_interpolation` defaults to `linear`, which is the only supported scheme for now.

Bonds section tag version	Tag attributes and default values	Required parameter attributes	Optional parameter attributes
0.3	`potential="harmonic"`, `fractional_bondorder_method="none"`, `fractional_bondorder_interpolation="linear"`	`smirks`, `length`, `k`	`id`, `parent_id`
0.4	`potential="(k/2)*(r-length)^2"`, `fractional_bondorder_method="AM1-Wiberg"`, `fractional_bondorder_interpolation="linear"`	`smirks`, `length`, `k`	`id`, `parent_id`

7.9.4 <Angles>

Angle parameters are specified via an <Angles>...</Angles> block, with individual <Angle> tags containing attributes specifying the equilibrium angle (angle) and force constant (k), as in this example:

```
<Angles version="0.3" potential="harmonic">
   <Angle smirks="[a,A:1]-[#6X4:2]-[a,A:3]" angle="109.50*degree" k="100.0*kilocalories_per_
↪mole/radian**2"/>
   <Angle smirks="[#1:1]-[#6X4:2]-[#1:3]" angle="109.50*degree" k="70.0*kilocalories_per_
↪mole/radian**2"/>
   ...
</Angles>
```

Currently, only potential="harmonic" is supported, where we utilize the standard harmonic functional form:

```
U(r) = (k/2)*(theta-angle)^2
```

Later revisions will add support for additional potential types and the ability to support arbitrary algebraic functional forms. If the potential attribute is omitted, it defaults to harmonic.

Note that AMBER and CHARMM define a modified functional form, such that U(r) = k*(theta-angle)^2, so that force constants would need to be multiplied by two in order to be used in the SMIRNOFF format.

Angles section tag version	Tag attributes and default values	Required parameter attributes	Optional parameter attributes
0.3	potential="harmonic"	smirks, angle, k	id, parent_id

7.9.5 <ProperTorsions>

Proper torsions are specified via a <ProperTorsions>...</ProperTorsions> block, with individual <Proper> tags containing attributes specifying the periodicity (periodicity#), phase (phase#), and barrier height (k#).

```
<ProperTorsions version="0.3" potential="k*(1+cos(periodicity*theta-phase))">
   <Proper smirks="[a,A:1]-[#6X4:2]-[#6X4:3]-[a,A:4]" idivf1="9" periodicity1="3" phase1="0.
↪0*degree" k1="1.40*kilocalories_per_mole"/>
   <Proper smirks="[#6X4:1]-[#6X4:2]-[#8X2:3]-[#6X4:4]" idivf1="1" periodicity1="3" phase1=
↪"0.0*degree" k1="0.383*kilocalories_per_mole" idivf2="1" periodicity2="2" phase2="180.
↪0*degree" k2="0.1*kilocalories_per_mole"/>
   ...
</ProperTorsions>
```

Here, child Proper tags specify at least k1, phase1, and periodicity1 attributes for the corresponding parameters of the first force term applied to this torsion. However, additional values are allowed in the form k#, phase#, and periodicity#, where all # values must be consecutive (e.g., it is impermissible to specify k1 and k3 values without a k2 value) but # can go as high as necessary.

For convenience, and optional attribute specifies a torsion multiplicity by which the barrier height should be divided (idivf#). The default behavior of this attribute can be controlled by the top-level attribute default_idivf (default: "auto") for <ProperTorsions>, which can be an integer (such as "1") controlling the value of idivf if not specified or "auto" if the barrier height should be divided by the number of torsions impinging on the central bond. For example:

```
<ProperTorsions version="0.3" potential="k*(1+cos(periodicity*theta-phase))" default_idivf=
↪"auto">
   <Proper smirks="[a,A:1]-[#6X4:2]-[#6X4:3]-[a,A:4]" periodicity1="3" phase1="0.0*degree"␣
↪k1="1.40*kilocalories_per_mole"/>
   ...
</ProperTorsions>
```

Currently, only `potential="k*(1+cos(periodicity*theta-phase))"` is supported, where we utilize the functional form:

```
U = \sum_{i=1}^N k_i * (1 + cos(periodicity_i * phi - phase_i))
```

Note: **AMBER defines a modified functional form**, such that `U = \sum_{i=1}^N (k_i/2) * (1 + cos(periodicity_i * phi - phase_i))`, so that barrier heights would need to be divided by two in order to be used in the SMIRNOFF format.

If the `potential` attribute is omitted, it defaults to `k*(1+cos(periodicity*theta-phase))`.

Fractional torsion bond orders

Fractional torsion bond orders can be used to allow interpolation and extrapolation of torsion parameters. This is similar to the functionality provided by fractional bond orders detailed above. For example, these parameters:

```
<ProperTorsions version="0.3" potential="k*(1+cos(periodicity*theta-phase))" default_idivf=
↪"auto">
    <Proper smirks="[*:1]:[#6X4:2]-[#6X4:3]:[*:4]" periodicity1="2" phase1="0.0 * degree" k1=
↪"1.00*kilocalories_per_mole" idivf1="1.0"/>
    <Proper smirks="[*:1]:[#6X4:2]=[#6X4:3]:[*:4]" periodicity1="2" phase1="0.0 * degree" k1=
↪"1.80*kilocalories_per_mole" idivf1="1.0"/>
    ...
```

can be replaced by a single parameter line by first defining the `fractional_bondorder_method` header-level attribute to specify a method for computing the fractional bond order and `fractional_bondorder_interpolation` for specifying the procedure for interpolating parameters between specified integer bond orders:

```
<ProperTorsions version="0.3" potential="k*(1+cos(periodicity*theta-phase))" default_idivf=
↪"auto" fractional_bondorder_method="AM1-Wiberg" fractional_bondorder_interpolation="linear
↪">
    <Proper smirks="[*:1]:[#6X4:2]~[#6X4:3]:[*:4]" periodicity1="2" phase1="0.0 * degree" k1_
↪bondorder1="1.00*kilocalories_per_mole" k1_bondorder2="1.80*kilocalories_per_mole" idivf1=
↪"1.0"/>
    ...
```

This allows specification of the barrier height for e.g. bond orders 1 and 2 (single and double bonds), and then interpolation between those based on the partial/fractional bond order. Note that in actual usage partial/fractional bond order may never be exactly 1 or 2, or perhaps even near 2; these values only serve to define the slope of the line used for interpolation. In the example above, we replaced the two proper torsion terms (one single central bond (-) and one double central bond (=) with a single term giving the barrier heights for bond order 1 and 2. If there are cases where the fractional bond order is 1.5, this can correspond

to e.g. an aromatic bond. When barrier heights for more than two integer bond orders are specified, (say, 1, 2, and 3), the interpolation lines are drawn between successive points as a piecewise linear function.

Cases in which the fractional bond order for the central bond is outside of the bond orders specified (e.g. 1 and 2 above), the barrier height `k#` is *extrapolated* using the same slope of the line used for interpolation. This works even when barrier heights for more than two integer bond orders are specified (say, 1, 2, and 3), in which case the piecewise linear extrapolation beyond the bounds uses the slope of the line defined by the nearest two bond orders. In other words, a fractional bond order of 3.2 would yield an interpolated `k#` value determined by the interpolation line between `k#_bondorder2` and `k#_bondorder3`. A fractional bond order of .9 would yield an interpolated `k#` value determined by the interpolation line between `k#_bondorder1` and `k#_bondorder2`.

Some key usage points:

- `fractional_bondorder_method` defaults to `AM1-Wiberg`.
- `fractional_bondorder_interpolation` defaults to `linear`, which is the only supported scheme for now.

Proper-Torsions section tag version	Tag attributes and default values	Required parameter attributes	Optional parameter attributes
0.3	potential="k*(1+cos(periodicity*theta-phase))", default_idivf="auto"	smirks, k, phase, periodicity	idivf, id, parent_id
0.4	potential="k*(1+cos(periodicity*theta-phase))", default_idivf="auto", fractional_bondorder_method="AM1-Wiberg", fractional_bondorder_interpolation="linear"	smirks, (k OR k_bondorder), phase, periodicity	idivf, id, parent_id

7.9.6 <ImproperTorsions>

Improper torsions are specified via an `<ImproperTorsions>...</ImproperTorsions>` block, with individual `<Improper>` tags containing attributes that specify the same properties as `<ProperTorsions>`:

```
<ImproperTorsions version="0.3" potential="k*(1+cos(periodicity*theta-phase))">
    <Improper smirks="[*:1]~[#6X3:2](=[#7X2,#7X3+1:3])~[#7:4]" k1="10.5*kilocalories_per_mole
↪" periodicity1="2" phase1="180.*degree"/>
    ...
</ImproperTorsions>
```

Currently, only `potential="charmm"` is supported, where we utilize the functional form:

```
U = \sum_{i=1}^N k_i * (1 + cos(periodicity_i * phi - phase_i))
```

Note: **AMBER defines a modified functional form**, such that `U = \sum_{i=1}^N (k_i/2) * (1 + cos(periodicity_i * phi - phase_i))`, so that barrier heights would need to be divided by two in order to be used in the SMIRNOFF format.

If the `potential` attribute is omitted, it defaults to `charmm`.

The improper torsion energy is computed as the average over all three impropers (all with the same handedness) in a trefoil. This avoids the dependence on arbitrary atom orderings that occur in more traditional

typing engines such as those used in AMBER. The *second* atom in an improper (in the example above, the trivalent carbon) is the central atom in the trefoil.

ImproperTorsions section tag version	Tag attributes and default values	Required parameter attributes	Optional parameter attributes
0.3	potential="k*(1+cos(periodicity*theta-phase))", default_idivf="auto"	smirks, k, phase, periodicity	idivf, id, parent_id

7.9.7 <GBSA>

Warning: The current release of ParmEd can not transfer GBSA models produced by the Open Force Field Toolkit to other simulation packages. These GBSA forces are currently only computable using OpenMM.

Generalized-Born surface area (GBSA) implicit solvent parameters are optionally specified via a <GBSA>...</GBSA> using <Atom> tags with GBSA model specific attributes:

```
<GBSA version="0.3" gb_model="OBC1" solvent_dielectric="78.5" solute_dielectric="1" sa_model=
↪"ACE" surface_area_penalty="5.4*calories/mole/angstroms**2" solvent_radius="1.4*angstroms">
  <Atom smirks="[*:1]" radius="0.15*nanometer" scale="0.8"/>
  <Atom smirks="[#1:1]" radius="0.12*nanometer" scale="0.85"/>
  <Atom smirks="[#1:1]~[#7]" radius="0.13*nanometer" scale="0.85"/>
  <Atom smirks="[#6:1]" radius="0.17*nanometer" scale="0.72"/>
  <Atom smirks="[#7:1]" radius="0.155*nanometer" scale="0.79"/>
  <Atom smirks="[#8:1]" radius="0.15*nanometer" scale="0.85"/>
  <Atom smirks="[#9:1]" radius="0.15*nanometer" scale="0.88"/>
  <Atom smirks="[#14:1]" radius="0.21*nanometer" scale="0.8"/>
  <Atom smirks="[#15:1]" radius="0.185*nanometer" scale="0.86"/>
  <Atom smirks="[#16:1]" radius="0.18*nanometer" scale="0.96"/>
  <Atom smirks="[#17:1]" radius="0.17*nanometer" scale="0.8"/>
</GBSA>
```

Supported Generalized Born (GB) models

In the <GBSA> tag, gb_model selects which GB model is used. Currently, this can be selected from a subset of the GBSA models available in OpenMM:

- HCT : Hawkins-Cramer-Truhlar (corresponding to igb=1 in AMBER): requires parameters [radius, scale]
- OBC1 : Onufriev-Bashford-Case using the GB(OBC)I parameters (corresponding to igb=2 in AMBER): requires parameters [radius, scale]
- OBC2 : Onufriev-Bashford-Case using the GB(OBC)II parameters (corresponding to igb=5 in AMBER): requires parameters [radius, scale]

If the gb_model attribute is omitted, it defaults to OBC1.

The attributes solvent_dielectric and solute_dielectric specify solvent and solute dielectric constants used by the GB model. In this example, radius and scale are per-particle parameters of the OBC1 GB model supported by OpenMM.

Surface area (SA) penalty model

The `sa_model` attribute specifies the solvent-accessible surface area model ("SA" part of GBSA) if one should be included; if omitted, no SA term is included.

Currently, only the analytical continuum electrostatics (ACE) model, designated `ACE`, can be specified, but there are plans to add more models in the future, such as the Gaussian solvation energy component of EEF1. If `sa_model` is not specified, it defaults to `ACE`.

The `ACE` model permits two additional parameters to be specified:

- The `surface_area_penalty` attribute specifies the surface area penalty for the `ACE` model. (Default: `5.4 calories/mole/angstroms**2`)
- The `solvent_radius` attribute specifies the solvent radius. (Default: `1.4 angstroms`)

GBSA section tag version	Tag attributes and default values	Required parameter attributes	Optional parameter attributes
0.3	`gb_model="OBC1", solvent_dielectric="78.5", solute_dielectric="1", sa_model="ACE", surface_area_penalty="5.4*calories/mole/angstrom**2", solvent_radius="1.4*angstrom"`	`smirks, radius, scale`	`id, parent_id`

7.9.8 <Constraints>

Bond length or angle constraints can be specified through a `<Constraints>` block, which can constrain bonds to their equilibrium lengths or specify an interatomic constraint distance. Two atoms must be tagged in the `smirks` attribute of each `<Constraint>` record.

To constrain the separation between two atoms to their equilibrium bond length, it is critical that a `<Bonds>` record be specified for those atoms:

```
<Constraints version="0.3" >
  <!-- constrain all bonds to hydrogen to their equilibrium bond length -->
  <Constraint smirks="[#1:1]-[*:2]" />
</Constraints>
```

Note that the two atoms must be bonded in the specified `Topology` for the equilibrium bond length to be used.

To specify the constraint distance, or constrain two atoms that are not directly bonded (such as the hydrogens in rigid water models), specify the `distance` attribute (and optional `distance_unit` attribute for the `<Constraints>` tag):

```
<Constraints version="0.3">
  <!-- constrain water O-H bond to equilibrium bond length (overrides earlier constraint) -->
  <Constraint smirks="[#1:1]-[#8X2H2:2]-[#1]" distance="0.9572*angstrom"/>
  <!-- constrain water H...H, calculating equilibrium length from H-O-H equilibrium angle␣
↪and H-O equilibrium bond lengths -->
  <Constraint smirks="[#1:1]-[#8X2H2]-[#1:2]" distance="1.8532*angstrom"/>
</Constraints>
```

Typical molecular simulation practice is to constrain all bonds to hydrogen to their equilibrium bond lengths and enforce rigid TIP3P geometry on water molecules:

```
<Constraints version="0.3">
  <!-- constrain all bonds to hydrogen to their equilibrium bond length -->
  <Constraint smirks="[#1:1]-[*:2]" />
  <!-- TIP3P rigid water -->
  <Constraint smirks="[#1:1]-[#8X2H2:2]-[#1]" distance="0.9572*angstrom"/>
  <Constraint smirks="[#1:1]-[#8X2H2]-[#1:2]" distance="1.8532*angstrom"/>
</Constraints>
```

Constraint section tag version	Required tag attributes and default values	Required parameter attributes	Optional parameter attributes
0.3		`smirks`	`distance`

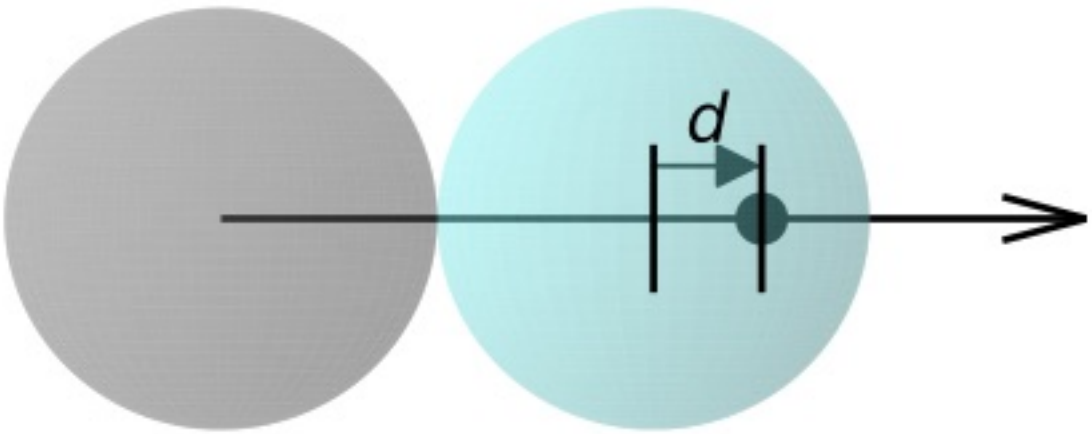
7.10 Advanced features

Standard usage is expected to rely primarily on the features documented above and potentially new features. However, some advanced features will also be supported.

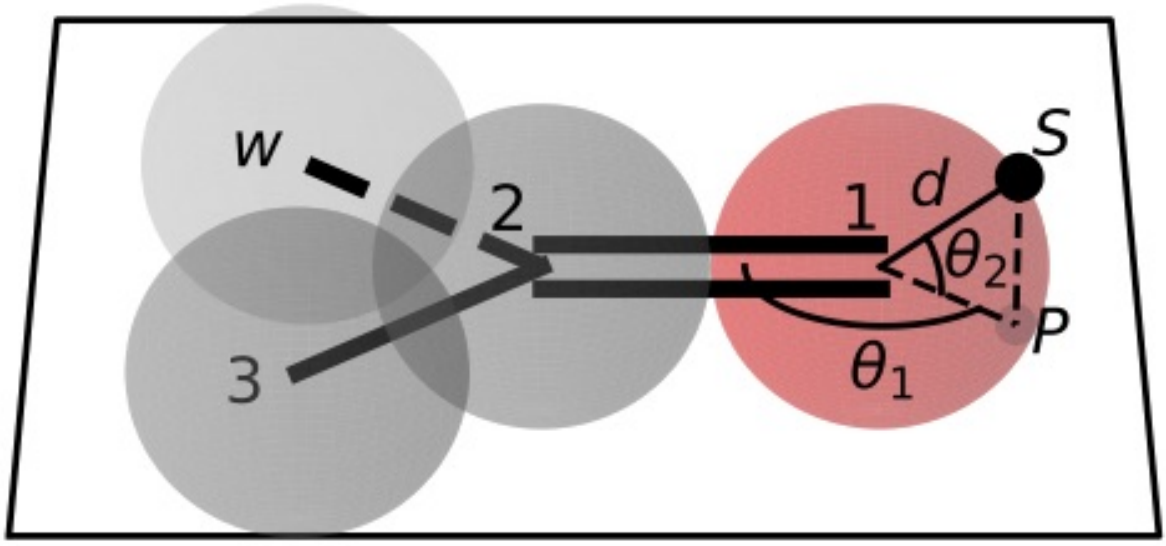
7.10.1 `<VirtualSites>`: Virtual sites for off-atom charges

We will implement experimental support for placement of off-atom (off-center) charges in a variety of contexts which may be chemically important in order to allow easy exploration of when these will be warranted. We will support the following different types or geometries of off-center charges (as diagrammed below):

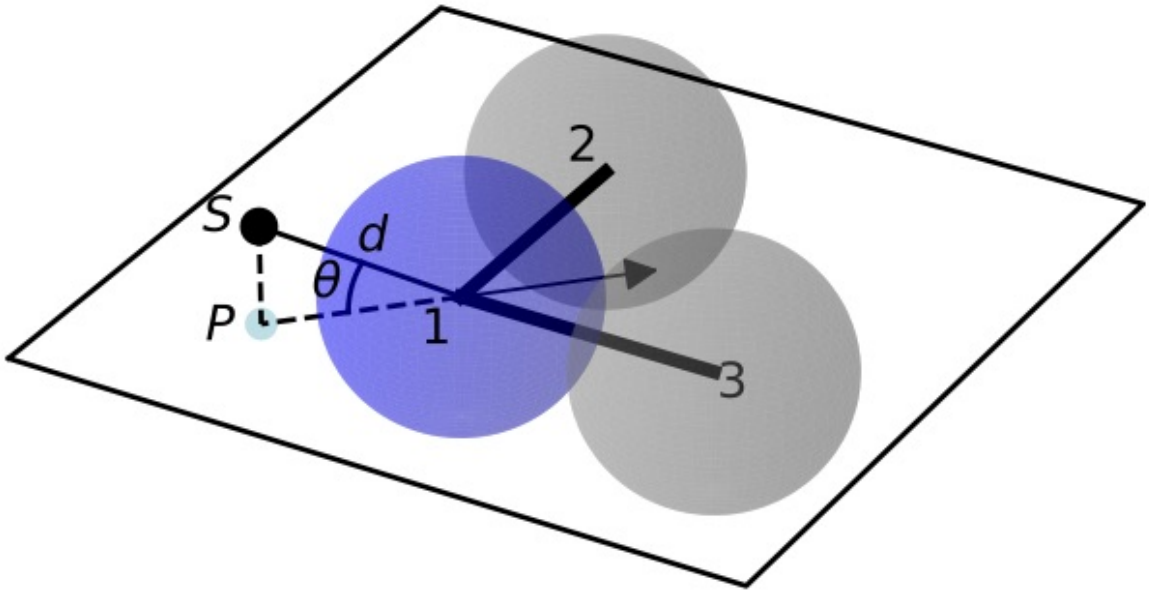
- `BondCharge`: This supports placement of a virtual site `S` along a vector between two specified atoms, e.g. to allow for a sigma hole for halogens or similar contexts. With positive values of the distance, the virtual site lies outside the first indexed atom (green in this image).



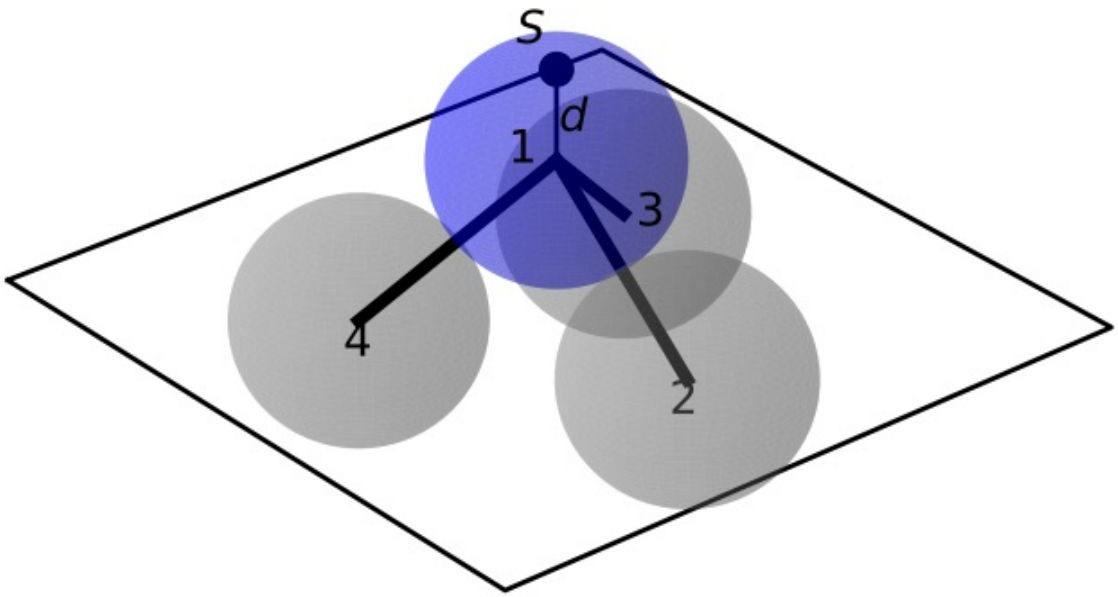
- `MonovalentLonePair`: This is originally intended for situations like a carbonyl, and allows placement of a virtual site `S` at a specified distance `d`, `inPlaneAngle` (theta 1 in the diagram), and `outOfPlaneAngle` (theta 2 in the diagram) relative to a central atom and two connected atoms.



- `DivalentLonePair`: This is suitable for cases like four-point and five-point water models as well as pyrimidine; a charge site `S` lies a specified distance `d` from the central atom among three atoms (blue) along the bisector of the angle between the atoms (if `outOfPlaneAngle` is zero) or out of the plane by the specified angle (if `outOfPlaneAngle` is nonzero) with its projection along the bisector. For positive values fo the distance `d` the virtual site lies outside the 2-1-3 angle and for negative values it lies inside.



- `TrivalentLonePair`: This is suitable for planar or tetrahedral nitrogen lone pairs; a charge site `S` lies above the central atom (e.g. nitrogen, blue) a distance `d` along the vector perpendicular to the plane of the three connected atoms (2,3,4). With positive values of `d` the site lies above the nitrogen and with negative values it lies below the nitrogen.



Each virtual site receives charge which is transferred from the desired atoms specified in the SMIRKS pattern via a `charge_increment#` parameter, e.g., if `charge_increment1=0.1*elementary_charge` then the virtual site will receive a charge of -0.1 and the atom labeled `1` will have its charge adjusted upwards by 0.1. N may index any indexed atom. Additionally, each virtual site can bear Lennard-Jones parameters, specified by `sigma` and `epsilon` or `rmin_half` and `epsilon`. If unspecified these default to zero.

In the SMIRNOFF format, these are encoded as:

```
<VirtualSites version="0.3" exclusion_policy="parents">
  <!-- sigma hole for halogens: "distance" denotes distance along the 2->1 bond vector,␣
↪measured from atom 2 -->
  <!-- Specify that 0.2 charge from atom 1 and 0.1 charge units from atom 2 are to be moved␣
↪to the virtual site, and a small Lennard-Jones site is to be added (sigma=0.1*angstroms,␣
↪epsilon=0.05*kcal/mol) -->
  <VirtualSite type="BondCharge" smirks="[Cl:1]-[C:2]" distance="0.30*angstrom" charge_
↪increment1="-0.2*elementary_charge" charge_increment2="-0.1*elementary_charge" sigma="0.
↪1*angstrom" epsilon="0.05*kilocalories_per_mole"/>
  <!-- Charge increments can extend out to as many atoms as are labeled, e.g. with a third␣
↪atom: -->
  <VirtualSite type="BondCharge" smirks="[Cl:1]-[C:2]~[*:3]" distance="0.30*angstrom" charge_
↪increment1="-0.1*elementary_charge" charge_increment2="-0.1*elementary_charge" charge_
↪increment3="-0.05*elementary_charge" sigma="0.1*angstrom" epsilon="0.05*kilocalories_per_
↪mole"/>
  <!-- monovalent lone pairs: carbonyl -->
  <!-- X denotes the charge site, and P denotes the projection of the charge site into the␣
↪plane of 1 and 2. -->
  <!-- inPlaneAngle is angle point P makes with 1 and 2, i.e. P-1-2 -->
  <!-- outOfPlaneAngle is angle charge site (X) makes out of the plane of 2-1-3 (and P)␣
↪measured from 1 -->
  <!-- Since unspecified here, sigma and epsilon for the virtual site default to zero -->
  <VirtualSite type="MonovalentLonePair" smirks="[O:1]=[C:2]-[*:3]" distance="0.30*angstrom"␣
↪outOfPlaneAngle="0*degree" inPlaneAngle="120*degree" charge_increment1="0.2*elementary_
↪charge" charge_increment2="0.2*elementary_charge" charge_increment3="0.2*elementary_charge
↪"/>
```

(continues on next page)

(continued from previous page)

```
  <!-- divalent lone pair: pyrimidine, TIP4P, TIP5P -->
  <!-- The atoms 2-1-3 define the X-Y plane, with Z perpendicular. If outOfPlaneAngle is 0,␣
↪the charge site is a specified distance along the in-plane vector which bisects the angle␣
↪left by taking 360 degrees minus angle(2,1,3). If outOfPlaneAngle is nonzero, the charge␣
↪sites lie out of the plane by the specified angle (at the specified distance) and their in-
↪plane projection lines along the angle's bisector. -->
  <VirtualSite type="DivalentLonePair" smirks="[*:2]~[#7X2:1]~[*:3]" distance="0.30*angstrom
↪" outOfPlaneAngle="0.0*degree" charge_increment1="0.1*elementary_charge" charge_increment2=
↪"0.2*elementary_charge" charge_increment3="0.2*elementary_charge"/>
  <!-- trivalent nitrogen lone pair -->
  <!-- charge sites lie above and below the nitrogen at specified distances from the␣
↪nitrogen, along the vector perpendicular to the plane of (2,3,4) that passes through the␣
↪nitrogen. If the nitrogen is co-planar with the connected atom, charge sites are simply␣
↪above and below the plane -->
  <!-- Positive and negative values refer to above or below the nitrogen as measured␣
↪relative to the plane of (2,3,4), i.e. below the nitrogen means nearer the 2,3,4 plane␣
↪unless they are co-planar -->
  <!-- To ensure that the second site does not overwrite the first, specify a unique name␣
↪for each. -->
  <VirtualSite type="TrivalentLonePair" smirks="[*:2]~[#7X3:1](~[*:4])~[*:3]" name="A"␣
↪distance="0.30*angstrom" charge_increment1="0.1*elementary_charge" charge_increment2="0.
↪2*elementary_charge" charge_increment3="0.2*elementary_charge" charge_increment4="0.
↪2*elementary_charge"/>
  <VirtualSite type="TrivalentLonePair" smirks="[*:2]~[#7X3:1](~[*:4])~[*:3]" name="B"␣
↪distance="-0.30*angstrom" charge_increment1="0.1*elementary_charge" charge_increment2="0.
↪2*elementary_charge" charge_increment3="0.2*elementary_charge" charge_increment4="0.
↪2*elementary_charge"/>
</VirtualSites>
```

VirtualSites section tag version	Tag attributes and default values	Required parameter attributes and default values	Optional parameter attributes
0.3	exclusion_policy="parents"	smirks, type, distance, charge_increment (indexed), inPlaneAngle IF type="MonovalentLonePair", outOfPlaneAngle IF type="MonovalentLonePair OR type="DivalentLonePair", sigma=0.*angstrom, epsilon=0.*kilocalories_per_mole, name="EP", match="all_permutations" IF type=BondCharge OR type="MonovalentLonePair OR type="DivalentLonePair", match="once" IF type="TrivalentLonePair	N/A

7.10.2 Aromaticity models

Before conducting SMIRKS substructure searches, molecules are prepared using one of the supported aromaticity models, which must be specified with the `aromaticity_model` attribute. The only aromaticity model currently widely supported (by both the OpenEye toolkit and RDKit) is the `OEAroModel_MDL` model.

7.10.3 Additional plans for future development

See the OpenFF toolkit GitHub issue tracker to propose changes to this specification, or read through proposed changes currently being discussed.

7.11 The `openforcefield` reference implementation

A Python reference implementation of a parameterization engine implementing the SMIRNOFF force field specification can be found online. This implementation can use either the free-for-academics (but commercially supported) OpenEye toolkit or the free and open source RDKit cheminformatics toolkit. See the installation instructions for information on how to install this implementation and its dependencies.

7.11.1 Examples

A relatively extensive set of examples is made available on the reference implementation repository under examples/.

7.11.2 Parameterizing a system

Consider parameterizing a simple system containing a the drug imatinib.

```
# Create a molecule from a mol2 file
from openff.toolkit.topology import Molecule
molecule = Molecule.from_file('imatinib.mol2')

# Create a Topology specifying the system to be parameterized containing just the molecule
topology = molecule.to_topology()

# Load the first release of the "Parsley" force field
from openff.toolkit.typing.engines.smirnoff import ForceField
forcefield = ForceField('openff-1.0.0.offxml')

# Create an OpenMM System from the topology
system = forcefield.create_openmm_system(topology)
```

See `examples/SMIRNOFF_simulation/` for an extension of this example illustrating how to simulate this molecule in the gas phase.

The `topology` object provided to `create_openmm_system()` can contain any number of molecules of different types, including biopolymers, ions, buffer molecules, or solvent molecules. The OpenFF toolkit provides a number of convenient methods for importing or constructing topologies given PDB files, Sybyl mol2 files, SDF files, SMILES strings, and IUPAC names; see the toolkit documentation for more information. Notably, this `topology` object differs from those found in OpenMM or MDTraj in that it contains information on the *chemical identity* of the molecules constituting the system, rather than this atomic elements and covalent

connectivity; this additional chemical information is required for the direct chemical perception features of SMIRNOFF typing.

7.11.3 Using SMIRNOFF small molecule force fields with traditional biopolymer force fields

While SMIRNOFF format force fields can cover a wide range of biological systems, our initial focus is on general small molecule force fields, meaning that users may have considerable interest in combining SMIRNOFF small molecule parameters to systems in combination with traditional biopolymer parameters from conventional force fields, such as the AMBER family of protein/nucleic acid force fields. Thus, we provide an example of setting up a mixed protein-ligand system in examples/using_smirnoff_with_amber_protein_forcefield, where an AMBER family force field is used for a protein and the original "Parsley" force field (`openff-1.0.0`) for a small molecule.

7.11.4 The optional `id` and `parent_id` attributes and other XML attributes

In general, additional optional XML attributes can be specified and will be ignored by `ForceField` unless they are specifically handled by the parser (and specified in this document).

One attribute we have found helpful in parameter file development is the `id` attribute for a specific parameter line, and we *recommend* that SMIRNOFF force fields utilize this as effectively a parameter serial number, such as in:

```
<Bond smirks="[#6X3:1]-[#6X3:2]" id="b5" k="820.0*kilocalorie_per_mole/angstrom**2" length=
↪"1.45*angstrom"/>
```

Some functionality in `ForceField`, such as `ForceField.label_molecules`, looks for the `id` attribute. Without this attribute, there is no way to uniquely identify a specific parameter line in the XML file without referring to it by its smirks string, and since some smirks strings can become long and relatively unwieldy (especially for torsions) this provides a more human- and search-friendly way of referring to specific sets of parameters.

The `parent_id` attribute is also frequently used to denote parameters from which the current parameter is derived in some manner.

7.11.5 A remark about parameter availability

`ForceField` will currently raise an exception if any parameters are missing where expected for your system—i.e. if a bond is assigned no parameters, an exception will be raised. However, use of generic parameters (i.e. `[*:1]~[*:2]` for a bond) in your `.offxml` will result in parameters being assigned everywhere, bypassing this exception. We recommend generics be used sparingly unless it is your intention to provide true universal generic parameters.

7.12 Version history

7.12.1 0.3

This is a backwards-incompatible update to the SMIRNOFF 0.2 draft specification. However, the Open Force Field Toolkit version accompanying this update is capable of converting 0.1 spec SMIRNOFF data to 0.2 spec, and subsequently 0.2 spec to 0.3 spec. The 0.1-to-0.2 spec conversion makes a number of assumptions about settings such as long-range nonbonded handling. Warnings are printed about each assumption that is made during this spec conversion. No mechanism to convert backwards in spec is provided.

Key changes in this version of the spec are:

- Section headers now contain individual versions, instead of relying on the `<SMIRNOFF>`-level tag.
- Section headers no longer contain `X_unit` attributes.
- All physical quantities are now written as expressions containing the appropriate units.
- The default potential for `<ProperTorsions>` and `<ImproperTorsions>` was changed from `charmm` to `k*(1+cos(periodicity*theta-phase))`, as CHARMM interprets torsion terms with perioidicity 0 as having a quadratic potential, while the Open Force Field Toolkit would interpret a zero periodicity literally.

7.12.2 0.2

This is a backwards-incompatible overhaul of the SMIRNOFF 0.1 draft specification along with `ForceField` implementation refactor:

- Aromaticity model now defaults to `OEAroModel_MDL`, and aromaticity model names drop OpenEye-specific prefixes
- Top-level tags are now required to specify units for any unit-bearing quantities to avoid the potential for mistakes from implied units.
- Potential energy component definitions were renamed to be more general:
  - `<NonbondedForce>` was renamed to `<vdW>`
  - `<HarmonicBondForce>` was renamed to `<Bonds>`
  - `<HarmonicAngleForce>` was renamed to `<Angles>`
  - `<BondChargeCorrections>` was renamed to `<ChargeIncrementModel>` and generalized to accommodate an arbitrary number of tagged atoms
  - `<GBSAForce>` was renamed to `<GBSA>`
- `<PeriodicTorsionForce>` was split into `<ProperTorsions>` and `<ImproperTorsions>`
- `<vdW>` now specifies 1-2, 1-3, 1-4, and 1-5 scaling factors via `scale12` (default: 0), `scale13` (default: 0), `scale14` (default: 0.5), and `scale15` (default 1.0) attributes. It also specifies the long-range vdW method to use, currently supporting `cutoff` (default) and `PME`. Coulomb scaling parameters have been removed from `StericsForce`.
- Added the `<Electrostatics>` tag to separately specify 1-2, 1-3, 1-4, and 1-5 scaling factors for electrostatics, as well as the method used to compute electrostatics (`PME`, `reaction-field`, `Coulomb`) since this has a huge effect on the energetics of the system.
- Made it clear that `<Constraint>` entries do not have to be between bonded atoms.

- <VirtualSites> has been added, and the specification of charge increments harmonized with <ChargeIncrementModel>
- The potential attribute was added to most forces to allow flexibility in extending forces to additional functional forms (or algebraic expressions) in the future. potential defaults to the current recommended scheme if omitted.
- <GBSA> now has defaults specified for gb_method and sa_method
- Changes to how fractional bond orders are handled:
  - Use of fractional bond order is now are specified at the force tag level, rather than the root level
  - The fractional bond order method is specified via the fractional_bondorder_method attribute
  - The fractional bond order interpolation scheme is specified via the fractional_bondorder_interpolation
- Section heading names were cleaned up.
- Example was updated to reflect use of the new openff.toolkit.topology.Topology class
- Eliminated "Requirements" section, since it specified requirements for the software, rather than described an aspect of the SMIRNOFF specification
- Fractional bond orders are described in <Bonds>, since they currently only apply to this term.

7.12.3 0.1

Initial draft specification.

CHAPTER

EIGHT

VIRTUAL SITES

The Open Force Field Toolkit supports the SMIRNOFF virtual site specification for models using off-site charges, including 4- and 5-point water models, in addition to lone pair modelling on various functional groups. The primary focus is on the ability to parameterize a system using virtual sites, and generate an OpenMM system with all virtual sites present and ready for evaluation. Support for formats other than OpenMM has not yet been implemented, but may come with the appearance of the OpenFF system object. In addition to implementing the specification, the toolkit `Molecule` object allows the creation and manipulation of virtual sites.

8.1 Support for the SMIRNOFF VirtualSite tag

Virtual sites can be added to a System in two ways:

- SMIRNOFF Force Fields can contain a VirtualSites tag, specifying the addition of virtual sites according to SMARTS-based rules.
- Virtual sites can be added to a `Molecule`, and these will appear in the final OpenMM system if a virtual site handler is present in the`ForceField`.

Virtual sites directly depend on 3D conformation, because the position of the virtual sites depends on vectors defined by the atoms specified during parameterization. Because of this, a virtual site matching the triplet of atoms 1-2-3 will define a point that is different from a triplet matching 3-2-1. This is similar to defining "right-handed" and "left-handed" coordinate systems. This subtlety plays into two major concepts in force field development:

1. We sometimes want to define a single virtual site describing two points with the same parameters (distance, angle, etc.) via symmetry, such as 5-point water models.
2. We have a match that produces multiple orderings of the atoms (e.g. if wildcards are present in the SMARTS pattern), and we only want one to be applied.

Case (1) is very useful for parameter optimization, where a single SMARTS-based parameter can be used to optimize both points, such as the angle defining the virtual points for a 5-point water model. Case (2) is the typical scenario for the nitrogen lone pair in ammonia, where only one point needs to be specified. We discuss a few more illustrative examples below. Beyond these attributes, the virtual site specification allows a policy for specifying how to handle exclusions in the OpenMM force evaluator. The current default is to add pairwise energy exclusions in the OpenMM system between a virtual site and all tagged atoms matched in its SMARTS(`exclusion_policy="parents"`,). Currently defined are `"none"`, `"minimal"`, and `"parents"`, where `"minimal"` specifies the single atom that the virtual site defines as the "origin". For water, for example, `"minimal"` would mean just the oxygen, whereas `"parents"` would mean all three atoms.

In order to give consistent and intended behavior, the specification was modified from its draft form in following manner: The `"name"` and `"match"` attributes have been added to each virtual site parameter type. These changes allow for

- specifying different virtual site types using the same atoms
- allowing two virtual sites with the same type and same atoms but different physical parameters to be added simultaneously
- allowing the ability to control whether the virtual site encodes one or multiple particles, based on the number of ways the matching atoms can be ordered.

The `"name"` attribute encodes whether the virtual site to be added should override an existing virtual site of the same type (e.g. hierarchy preference), or if this virtual site should be added in addition to the other existing virtual sites on the given atoms. This means that different virtual site types can share the same group of parent atoms and use the same name without overwriting each other (the default `name` is `EP` for all sites, which gives the expected hierarchical behavior used in other SMIRNOFF tags).

The `"match"` attribute accepts either `"once"` or `"all_permutations"`, offering control for situations where a SMARTS pattern can possibly match the same group of atoms in different orders(either due to wildcards or local symmetry) and it is desired to either add just one or all of the possible virtual particles. The default value is `"all_permutations"`, but for `TrivalentLonePair` it is always set to `"once"`, regardless of what the file contains, since all orderings always place the particle in the exact same position.

The following cases exemplify our reasoning in implementing this behavior, and should draw caution to complex issues that may arise when designing virtual site parameters. Let us consider 4-, 5-, and 6-point water models:

- A 4-point water model with a `DivalentLonePair`: This can be implemented by specifying `match="once"`, `outOfPlaneAngle="0*degree"`, and `distance=-.15*angstrom"`. Since the SMIRKS pattern `"[#1:1]-[#8X2:2]- [#2:3]"` would match water twice and would create two particles in the exact same position if `all_permutations` was specified, we specify `"once"` to have only one particle generated. Although having two particles in the same position should not affect the physics if the proper exclusion policy is applied, it would effectively make the 4-point model just as expensive as 5-point models.
- A 5-point water model with a `DivalentLonePair`: This can be implemented by using `match="all_permutations"` (unlike the 4-point model), `outOfPlaneAngle="56.26*degree`, and `distance=0.7*angstrom`, for example. Here the permutations will cause particles to be placed at $\pm$56.26 degrees, and changing any of the physical quantities will affect *both* particles.
- A 6-point water model with both `DivalentLonePair` sites above. Since these two parameters look identical, it is unclear whether they should both be applied or if one should override the other. The toolkit never compares the physical numbers to determine equality as this can lead to instability during e.g. parameter fitting. To get this to work, we specify `name="EP1"` for the first parameter, and `name="EP2"` for the second parameter. This instructs the parameter handler keep them separate, and therefore both are applied. (If both had the same name, then the typical SMIRNOFF hierarchy rules are used, and only the last matched parameter would be applied.)
- Dinitrogen, `N#N` with a `BondCharge` virtual site. Since we want a `BondCharge` on both ends, we specify `match="all_permutations"`.
- Formaldehyde, `H2C=O`, with `MonovalentLonePair` virtual site(s) on the oxygen, with the aim of modeling both lone pairs. This one is subtle, since `[#1:3]-[#6X3:2]=[#8X1:1]` matches two unique groups of atoms (`1-3-4` and `2-3-4`). It is important to note in this situation that `match="all_permutations"` behaves exactly the same as `match="once"`. Due to the anchoring hydrogens (`1` and `2`) being symmetric but opposite about the bond between `3` and `4`, a single parameter does correctly place both lone pairs. A standing issue here is that the default exclusion policy (`parents`) will allow these two virtual sites to interact since they have different indexed atoms (parents), causing the energy to be different than the non-virtual site parameterization. In the future, the `exclusion_policy="local"` will account for this, and make virtual sites that share at least one "parent" atom not interact with each other. As a special note: when applying a `MonovalentLonePair` to a completely symmetric molecule, e.g. water, `all_permutations` can come into play, but this will apply two particles (one for each hydrogen).

Finally, the toolkit handles the organization of atoms and virtual sites in a specific manner. Virtual sites are expected to be added *after all molecules in the topology are present*. This is because the Open Force Field Toolkit organizes a topology by placing all atoms first, then all virtual sites last. This differs from the OpenMM Modeller object, for example, which interleaves the order of atoms and virtual sites in such a way that all particles of a molecule are contiguous. In addition, due to the fact that a virtual site may contain multiple particles coupled to single parameters, the toolkit makes a distinction between a virtual *site*, and a virtual *particle*. A virtual site may represent multiple virtual particles, so the total number of particles cannot be directly determined by simply summing the number of atoms and virtual sites in a molecule. This is taken into account, however, and the `Molecule` and `Topology` classes both implement `particle` iterators.

CHAPTER

NINE

DEVELOPING FOR THE TOOLKIT

This guide is written with the understanding that our contributors are NOT professional software developers, but are instead computational chemistry trainees and professionals. With this in mind, we aim to use a minimum of bleeding-edge technology and alphabet soup, and we will define any potentially unfamiliar processes or technologies the first time they are mentioned. We enforce use of certain practices (tests, formatting, coverage analysis, documentation) primarily because they are worthwhile upfront investments in the long-term sustainability of this project. The resources allocated to this project will come and go, but we hope that following these practices will ensure that minimal developer time will maintain this software far into the future.

The process of contributing to the OpenFF Toolkit is more than just writing code. Before contributing, it is a very good idea to start a discussion on the Issue tracker about the functionality you'd like to add. This Issue discussion will help us decide with you where in the codebase it should go, any overlapping efforts with other developers, and what the user experience should be. Please note that the OpenFF Toolkit is intended to be used primarily as one piece of larger workflows, and that simplicity and reliability are two of our primary goals. Often, the cost/benefit of new features must be discussed, as a complex codebase is harder to maintain. When new functionality is added to the OpenFF Toolkit, it becomes our responsibility to maintain it, so it's important that we understand contributed code and are in a position to keep it up to date.

9.1 Overview

9.1.1 Philosophy

- The *core functionality* of the OpenFF Toolkit is to combine an Open Force Field `ForceField` and `Topology` to create an OpenMM `System`.
- An OpenMM `System` contains *everything* needed to compute the potential energy of a system, except the coordinates and (optionally) box vectors.
- The OpenFF toolkit employs a modular "plugin" architecture wherever possible, providing a standard interface for contributed features.

9.1.2 Terminology

For high-level toolkit concepts and terminology important for both development and use of the Toolkit, see the *core concepts page*.

SMIRNOFF and the OpenFF Toolkit

SMIRNOFF data A hierarchical data structure that complies with the *SMIRNOFF specification*. This can be serialized in many formats, including XML (OFFXML). The subsections in a SMIRNOFF data source generally correspond to one energy term in the functional form of a force field.

Cosmetic attribute Data in a SMIRNOFF data source that does not correspond to a known attribute. These have no functional effect, but several programs use the extensibility of the OFFXML format to define additional attributes for their own use, and their workflows require the OFF toolkit to process the files while retaining these keywords.

Development Infrastructure

Continuous Integration (CI) Tests that run frequently while the code is undergoing changes, ensuring that the codebase still installs and has the intended behavior. Currently, we use a service called GitHub Actions for this. CI jobs run every time a commit is made to the `master` branch of the `openff-toolkit` GitHub repository or in a PR opened against it. These runs start by booting virtual machines that mimic brand new Linux and macOS computers. They then follow build instructions (see the `.github/workflows/CI.yml` file) to install the toolkit. After installing the OpenFF Toolkit and its dependencies, these virtual machines run our test suite. If the tests all pass, the build "passes" (returns a green check mark on GitHub).

If all the tests for a specific change to the `master` branch return green, then we know that the change has not broken the toolkit's existing functionality. When proposing code changes, we ask that contributors open a Pull Request (PR) on GitHub to merge their changes into the `master` branch. When a pull

request is open, CI will run on the latest set of proposed changes and indicate whether they are safe to merge through status checks, summarized as a green check mark or red cross.

CodeCov An extension to our testing framework that reports the fraction of our source code lines that were run during the tests (our "code coverage"). This functionality is actually the combination of several components – GitHub Actions runners run the tests using `pytest` with the `pytest-cov` plugin, and then coverage reports are uploaded to CodeCov's website. This analysis is re-run alongside the rest of our CI, and a badge showing our coverage percentage is in the project README.

"Looks Good To Me" (LGTM) A service that analyzes the code in our repository for simple style and formatting issues. This service assigns a letter grade to the codebase, and a badge showing our LGTM report is in the project README.

ReadTheDocs (RTD) A service that compiles and renders the package's documentation (from the `docs/` folder). The documentation itself can be accessed from the ReadTheDocs badge in the README. It is compiled by RTD alongside the other CI checks, and the compiled documentation for a pull request can be viewed by clicking the "details" link after the status.

9.1.3 User Experience

One important aspect of how we make design decisions is by asking "who do we envision using this software, and what would they want it to do here?". There is a wide range of possible users, from non-chemists, to students/trainees, to expert computational medicinal chemists. We have decided to build functionality intended for use by *expert medicinal chemists*, and whenever possible, add fatal errors if the toolkit risks doing the wrong thing. So, for example, if a molecule is loaded with an odd ionization state, we assume that the user has input it this way intentionally.

This design philosophy inevitably has trade-offs — For example, the OpenFF Toolkit will give the user a hard time if they try to load a "dirty" molecule dataset, where some molecules have errors or are not described in enough detail for the toolkit to unambiguously parametrize them. If there is risk of misinterpreting the molecule (for example, bond orders being undefined or chiral centers without defined stereochemistry), the toolkit should raise an error that the user can override. In this regard we differ from RDKit, which is more permissive in the level of detail it requires when creating molecules. This makes sense for RDKit's use cases, as several of its analyses can operate with a lower level of detail about the molecules. Often, the same design decision is the best for all types of users, but when we do need to make trade-offs, we assume the user is an expert.

At the same time, we aim for "automagic" behavior whenever a decision will clearly go one way over another. System parameterization is an inherently complex topic, and the OFF toolkit would be nearly unusable if we required the user to explicitly approve every aspect of the process. For example, if a `Topology` has its `box_vectors` attribute defined, we assume that the resulting OpenMM `System` should be periodic.

9.2 Modular design features

There are a few areas where we've designed the toolkit with extensibility in mind. Adding functionality at these interfaces should be considerably easier than in other parts of the toolkit, and we encourage experimentation and contribution on these fronts.

These features have occasionally confusing names. "Parameter" here refers to a single value in a force field, as it is generally used in biophysics; it does not refer to an argument to a function. "Attribute" is used to refer to an XML attribute, which allows data to be defined for a particular tag; it does not refer to a member of a Python class or object. For example, in the following XML excerpt the `<SMIRNOFF>` tag has the attributes `version` and `aromaticity_model`:

```
<SMIRNOFF version="0.3" aromaticity_model="OEAroModel_MDL">
  ...
</SMIRNOFF>
```

"Member" is used here to describe Python attributes. This terminology is borrowed for the sake of clarity in this section from languages like C++ and Java.

9.2.1 ParameterAttribute

A `ParameterAttribute` is a single value that can be validated at runtime.

A `ParameterAttribute` can be instantiated as Python class or instance members to define the kinds of value that a particular parameter can take. They are used in the definitions of both `ParameterHandler` and `ParameterType`. The sorts of values a `ParameterAttribute` can take on are restricted by runtime validation. This validation is highly customizable, and may do things like allowing only certain values for a string or enforcing the correct units or array dimensions on the value; in fact, the validation can be defined using arbitrary code. The name of a `ParameterAttribute` should correspond exactly to the corresponding attribute in an OFFXML file.

IndexedParameterAttribute

An `IndexedParameterAttribute` is a `ParameterAttribute` with a sequence of values, rather than just one. Each value in the sequence is indexed by an integer.

The exact name of an `IndexedParameterAttribute` is NOT expected to appear verbatim in a OFFXML file, but instead should appear with a numerical integer suffix. For example the `IndexedParameterAttribute` `k` should only appear as `k1`, `k2`, `k3`, and so on in an OFFXML. The current implementation requires this indexing to start at 1 and subsequent values be contiguous (no skipping numbers), but does not enforce an upper limit on the integer.

For example, dihedral torsions are often parameterized as the sum of several sine wave terms. Each of the parameters of the sine wave `k`, `periodicity`, and `phase` is implemented as an `IndexedParameterAttribute`.

MappedParameterAttribute

A `MappedParameterAttribute` is a `ParameterAttribute` with several values, with some arbitrary mapping to access values.

IndexedMappedParameterAttribute

An `IndexedMappedParameterAttribute` is a `ParameterAttribute` with a sequence of maps of values.

9.2.2 ParameterHandler

ParameterHandler is a generic base class for objects that perform parameterization for one section in a SMIRNOFF data source. A ParameterHandler has the ability to produce one component of an OpenMM System. Extend this class to add a support for a new force or energy term to the toolkit.

Each ParameterHandler-derived class MUST implement the following methods and define the following attributes:

- create_force(self, system, topology, **kwargs): takes an OpenMM System and a OpenFF Topology as input, as well as optional keyword arguments, and modifies the System to contain the appropriate parameters.
- Class members ParameterAttributes: These correspond to the header-level attributes in a SMIRNOFF data source. For example, the Bonds tag in the SMIRNOFF spec has an optional fractional_bondorder_method field, which corresponds to the line fractional_bondorder_method = ParameterAttribute(default=None) in the BondHandler class definition. The ParameterAttribute and IndexedParameterAttribute classes offer considerable flexibility for validating inputs. Defining these attributes at the class level implements the corresponding behavior in the default __init__ function.
- Class members _MIN_SUPPORTED_SECTION_VERSION and _MAX_SUPPORTED_SECTION_VERSION. ParameterHandler versions allow us to evolve ParameterHandler behavior in a controlled, recorded way. Force field development is experimental by nature, and it is unlikely that the initial choice of header attributes is suitable for all use cases. Recording the "versions" of a SMIRNOFF spec tag allows us to encode the default behavior and API of a specific generation of a ParameterHandler, while allowing the safe addition of new attributes and behaviors. If these attributes are not defined, defaults in the base class will apply and updates introducing new versions may break the existing code.

Each ParameterHandler-derived class MAY implement:

- _KWARGS: Keyword arguments passed to ForceField.create_openmm_system are validated against the _KWARGS lists of each ParameterHandler that the ForceField owns. If present, these keyword arguments and their values will be passed on to the ParameterHandler.
- _TAGNAME: The name of the SMIRNOFF OFFXML tag used to parameterize the class. This tag should appear in the top level within the <SMIRNOFF> tag; see the Parameter generators section of the SMIRNOFF specification.
- _INFOTYPE: The ParameterType subclass used to parse the elements in the ParameterHandler's parameter list.
- _DEPENDENCIES: A list of ParameterHandler subclasses that, when present, must run before this one. Note that this is *not* a list of ParameterHandler subclasses that are required by this one. Ideally, child classes of ParameterHandler are entirely independent, and energy components of a force field form distinct terms; when this is impossible, _DEPENDENCIES may be used to guarantee execution order.
- to_dict: converts the ParameterHandler to a hierarchical dict compliant with the SMIRNOFF specification. The default implementation of this function should suffice for most developers.
- check_handler_compatibility: Checks whether this ParameterHandler is "compatible" with another. This function is used when a ForceField is attempted to be constructed from *multiple* SMIRNOFF data sources, and it is necessary to check that two sections with the same tag name can be combined in a sane way. For example, if the user instructed two vdW sections to be read, but the sections defined different vdW potentials, then this function should raise an Exception indicating that there is no safe way to combine the parameters. The default implementation of this function should suffice for most developers.
- postprocess_system: operates identically to create_force, but is run after each ParameterHandlers' create_force has already been called. The default implementation of this method simply does nothing, and should suffice for most developers.

9.2.3 ParameterType

ParameterType is a base class for the SMIRKS-based parameters of a ParameterHandler. Extend this alongside ParameterHandler to define and validate the force field parameters of a new force. This is analogous to ParmEd's XType classes, like BondType. A ParameterType should correspond to a single SMARTS-based parameter.

For example, the Lennard-Jones potential can be parameterized through either the size ParameterAttribute sigma or r_min, alongside the energy ParameterAttribute epsilon. Both options are handled through the vdWType class, a subclass of ParameterType.

9.2.4 Non-bonded methods as implemented in OpenMM

The SMIRNOFF specification describes the contents of a force field, which can be implemented in a number of different ways in different molecular simulation engines. The OpenMM implementation provided by the OpenFF Toolkit either produces an openmm.System containing a openmm.NonbondedForce object or raises an exception depending on how the non-bonded parameters are specified. Exceptions are raised when parameters are incompatible with OpenMM (IncompatibleParameterError) or otherwise against spec (SMIRNOFFSpecError), and also when they are appropriate for the spec but not yet implemented in the toolkit (SMIRNOFFSpecUnimplementedError). This table describes which NonbondedMethod is used in the produced NonbondedForce, or else which exception is raised.

vdw_method	electrostatics_method	periodic	OpenMM Nonbonded method or exception	Common case
cutoff	Coulomb	True	raises IncompatibleParameterError	
cutoff	Coulomb	False	openmm.NonbondedForce.NoCutoff	
cutoff	reaction-field	True	raises SMIRNOFFSpecUnimplementedError	
cutoff	reaction-field	False	raises SMIRNOFFSpecError	
cutoff	PME	True	openmm.NonbondedForce.PME	*
cutoff	PME	False	openmm.NonbondedForce.NoCutoff	
LJPME	Coulomb	True	raises IncompatibleParameterError	
LJPME	Coulomb	False	openmm.NonbondedForce.NoCutoff	
LJPME	reaction-field	True	raises IncompatibleParameterError	
LJPME	reaction-field	False	raises SMIRNOFFSpecError	
LJPME	PME	True	openmm.NonbondedForce.LJPME	
LJPME	PME	False	openmm.NonbondedForce.NoCutoff	

Notes:

- The most commonly-used case (including the Parsley line) is in the fifth row (cutoff vdW, PME electrostatics, periodic topology) and marked with an asterisk.
- For all cases included a non-periodic topology, openmm.NonbondedForce.NoCutoff is currently used.

- Electrostatics method `reaction-field` can only apply to periodic systems, however it is not currently implemented.
- LJPME (particle mesh ewald for LJ/vdW interactions) is not yet fully described in the SMIRNOFF specification.
- In the future, the OpenFF Toolkit may create multiple `CustomNonbondedForce` objects in order to better de-couple vdW and electrostatic interactions.

9.3 Contributing

We always welcome GitHub pull requests. For bug fixes, major feature additions, or refactoring, please raise an issue on the GitHub issue tracker first to ensure the design will be amenable to current developer plans. Development of new toolkit features generally proceeds in the following stages:

- Begin a discussion on the GitHub issue tracker to determine big-picture "what should this feature do?" and "does it fit in the scope of the OpenFF Toolkit?"
  - "... typically, for existing water models, we want to assign library charges"
- Start identifying details of the implementation that will be clear from the outset
  - "Create a new "special section" in the SMIRNOFF format (kind of analogous to the BondChargeCorrections section) which allows SMIRKS patterns to specify use of library charges for specific groups
  - "Following #86, here's how library charges might work: ..."
- Create a branch or fork for development
  - The OpenFF Toolkit has one unusual aspect of its CI build process, which is that certain functionality requires the OpenEye toolkits, so the builds must contain a valid OpenEye license file. An OpenEye license is stored as an encrypted token within the `openforcefield` organization on GitHub. For security reasons, builds run from forks cannot access this key. Therefore, tests that depend on the OpenEye Toolkits will be skipped on forks. Contributions run on forks are still welcome, especially as features that do not interact directly with the OpenEye Toolktis are not likely affected by this limitation.

9.3.1 Setting up a development environment

1. Install the `conda` package manager as part of the Anaconda Distribution from here
2. Set up conda environment:

```
git clone https://github.com/openforcefield/openff-toolkit
cd openff-toolkit/
# Create a conda environment with dependencies from env/YAML file
conda env create -n openff-dev -f devtools/conda-envs/test_env.yaml
conda activate openff-dev
# Perform editable/dirty dev install
pip install -e .
```

3. Obtain and store Open Eye license somewhere like `~/.oe_license.txt`. Optionally store the path in environmental variable `OE_LICENSE`, i.e. using a command like `echo "export OE_LICENSE=/Users/yournamehere/.oe_license.txt" >> ~/.bashrc`

Building the Docs

The documentation is composed of two parts, a hand-written user guide and an auto-generated API reference. Both are compiled by Sphinx, and can be automatically served and regenerated on changes with `sphinx-autobuild`. Documentation for released versions is available at ReadTheDocs. ReadTheDocs also builds the documentation for each Pull Request opened on GitHub and keeps the output for 90 days.

To add the documentation dependencies to your existing `openff-dev` Conda environment:

```
# Add the documentation requirements to your Conda environment
conda env update --name openff-dev --file docs/environment.yml
conda install --name openff-dev -c conda-forge sphinx-autobuild
```

To build the documentation from scratch:

```
# Build the documentation
# From the openff-toolkit root directory
conda activate openff-dev
cd docs
make html
# Documentation can be found in docs/_build/html/index.html
```

To watch the source directory for changes and automatically rebuild the documentation and refresh your browser:

```
# Host the docs on a local HTTP server and rebuild when a source file is changed
# Works best when the docs have already been built
# From the openff-toolkit root directory
conda activate openff-dev
sphinx-autobuild docs docs/_build/html --watch openff
# Then navigate your web browser to http://localhost:8000
```

9.3.2 Style guide

Development for the `openff-toolkit` conforms to the recommendations given by the Software Development Best Practices for Computational Chemistry guide.

The naming conventions of classes, functions, and variables follows PEP8, consistently with the best practices guide. The naming conventions used in this library not covered by PEP8 are:

- Use `file_path`, `file_name`, and `file_stem` to indicate `path/to/stem.extension`, `stem.extension`, and `stem` respectively, consistent with the variables in the `pathlib` module of the standard library.
- Use `n_x` to abbreviate "number of x" (e.g. `n_atoms`, `n_molecules`).

We place a high priority on code cleanliness and readability, even if code could be written more compactly. For example, 15-character variable names are fine. Triply nested list comprehensions are not.

The `openff-toolkit` has adopted code formatting tools ("linters") to maintain consistent style and remove the burden of adhering to these standards by hand. Currently, two are employed:

1. Black, the uncompromising code formatter, automatically formats code with a consistent style.
2. isort, sorts imports

There is a step in CI that uses these tools to check for a consistent style (see the file `.github/workflows/lint.yml`). These checks will use the most recent versions of each linter. To ensure that changes follow these standards, you can install and run these tools locally:

```
conda install black isort -c conda-forge
black openff
isort openff
```

Anything not covered above is currently up to personal preference, but this may change as new linters are added.

9.3.3 Pre-commit

The `pre-commit` tool can *optionally* be used to automate some or all of the above style checks. It automatically runs other programs ("hooks") when you run `git commit`. It aborts the commit with an exit code if any of the hooks fail, i.e. if `black` reformats code. This project uses `pre-commit ci`, a free service that enforces style on GitHub using the `pre-commit` framework in CI.

To use `pre-commit` locally, first install it:

```
conda conda install pre-commit -c conda-forge  # also available via pip
```

Then, install the pre-commit hooks (note that it installs the linters into an isolated virtual environment, not the current conda environment):

```
pre-commit install
```

Hooks will now run automatically before commits. Once installed, they should run in a total of a few seconds.

If `pre-commit` is not used by the developer and style issues are found, a `pre-commit.ci` bot may commit directly to a PR to make these fixes. This bot should only ever alter styl and never make functional changes to code.

Note that tests (too slow) and type-checking (weird reasons) are not run by `pre-commit`. You should still manually run tests before commiting code.

9.4 Supported Python versions

The OpenFF Toolkit roughly follows NEP 29. As of version 0.9.1 (March 2021) this means Python 3.7-3.9 is officially supported. We develop, test, and distribute on macOS and Linux-based operating systems. We do not currently support Windows. Some CI builds run using only RDKit as a backend, some run using only OpenEye Toolkits, and some run using both installed at once.

The CI matrix is currently as follows:

	Linux			macOS		
	RDKit	OpenEye	RDKit + OE	RDKit	OpenEye	RDKit + OE
Python 3.6 and older	No support after 0.9.1 (March 2021)					
Python 3.7	**Test**	**Test**	**Test**	**Test**	**Test**	**Test**
Python 3.8	**Test**	Skip	Skip	**Test**	Skip	Skip
Python 3.9	**Test**	Skip	Skip	**Test**	Skip	Skip
Python 3.10 and newer	Waiting on official releases and upstream support					

CHAPTER

TEN

MOLECULAR TOPOLOGY REPRESENTATIONS

This module provides pure-Python classes for representing molecules and molecular systems. These classes offer several advantages over corresponding `Topology` objects in OpenMM and MDTraj, including offering serialization to a variety of standard formats (including XML, JSON, YAML, BSON, TOML, and MessagePack).

10.1 Primary objects

—

10.2 Secondary objects

—

CHAPTER

ELEVEN

FORCE FIELD TYPING TOOLS

11.1 Chemical environments

Tools for representing and operating on chemical environments

—

11.2 Force field typing engines

Engines for applying parameters to chemical systems

11.2.1 The SMIRks-Native Open Force Field (SMIRNOFF)

A reference implementation of the SMIRNOFF specification for parameterizing biomolecular systems

ForceField

The `ForceField` class is a primary part of the top-level toolkit API. `ForceField` objects are initialized from SMIRNOFF data sources (e.g. an `OFFXML` file). For a basic example of OpenMM `System` creation using a `ForceField`, see `examples/SMIRNOFF_simulation`.

—

Parameter Type

`ParameterType` objects are representations of individual SMIRKS-based SMIRNOFF parameters. These are usually initialized during `ForceField` creation, and can be inspected and modified by users via the Python API. For more information, see `examples/forcefield_modification`.

—

Parameter Handlers

Each `ForceField` primarily consists of several `ParameterHandler` objects, which each contain the machinery to add one energy component to an OpenMM `System`. During `System` creation, each `ParameterHandler` registered to a `ForceField` has its `assign_parameters()` function called.

Parameter I/O Handlers

`ParameterIOHandler` objects handle reading and writing of serialzied SMIRNOFF data sources.

Parameter Attributes

`ParameterAttribute` and `IndexedParameterAttribute` provide a standard backend for ParameterHandler and Parameter attributes, while also enforcing validation of types and units.

CHAPTER

TWELVE

UTILITIES

12.1 Toolkit wrappers

The toolkit wrappers provide a simple uniform API for accessing minimal functionality of cheminformatics toolkits.

These toolkit wrappers are generally used through a `ToolkitRegistry`, which can be constructed with a desired precedence of toolkits:

```
>>> from openff.toolkit.utils.toolkits import ToolkitRegistry, OpenEyeToolkitWrapper,␣
↪RDKitToolkitWrapper, AmberToolsToolkitWrapper
>>> toolkit_registry = ToolkitRegistry()
>>> toolkit_precedence = [OpenEyeToolkitWrapper, RDKitToolkitWrapper,␣
↪AmberToolsToolkitWrapper]
>>> [ toolkit_registry.register_toolkit(toolkit) for toolkit in toolkit_precedence if␣
↪toolkit.is_available() ]
[None, None, None]
```

The toolkit wrappers can then be accessed through the registry:

```
>>> from openff.toolkit.utils.toolkits import GLOBAL_TOOLKIT_REGISTRY as toolkit_registry
>>> from openff.toolkit.topology.molecule import Molecule
>>> molecule = Molecule.from_smiles('Cc1ccccc1')
>>> smiles = toolkit_registry.call('to_smiles', molecule)
```

The order of toolkits, as specified in `toolkit_precedence` above, determines the order in which the called method is resolved, i.e. if the toolkit with highest precedence has a `to_smiles` method, that is the toolkit that will be called. If the toolkit with highest precedence does not have such a method, it is attempted with other toolkits until one is found. By default, if a toolkit with an appropriately-named method raises an exception of any type, then iteration over the registered toolkits stops and that exception is raised. To continue iteration if specific exceptions are encountered, customize this behavior using the optional `raise_exception_types` keyword argument to `ToolkitRegistry.call`. If no registered toolkits have the method, a ValueError is raised, containing a message listing the registered toolkits and exceptions (if any) that were ignored.

Alternatively, the global toolkit registry (which will attempt to register any available toolkits) can be used:

```
>>> from openff.toolkit.utils.toolkits import GLOBAL_TOOLKIT_REGISTRY as toolkit_registry
>>> len(toolkit_registry.registered_toolkits)
4
```

Individual toolkits can be registered or deregistered to control the backend that ToolkitRegistry calls resolve to. This can be useful for debugging and exploring subtley different behavior between toolkit wrappers.

```
from openff.toolkit.utils.toolkits import OpenEyeToolkitWrapper, BuiltInToolkitWrapper
from openff.toolkit.utils.toolkits import GLOBAL_TOOLKIT_REGISTRY as toolkit_registry
toolkit_registry.deregister_toolkit(OpenEyeToolkitWrapper)
toolkit_registry.register_toolkit(BuiltInToolkitWrapper)
toolkit_registry.registered_toolkits
```

For example, differences in `to_smiles` functionality between OpenEye toolkits and The RDKit can be explored by selecting which toolkit(s) are and are not registered.

```
>>> from openff.toolkit.utils.toolkits import OpenEyeToolkitWrapper, GLOBAL_TOOLKIT_REGISTRY␣
↪as toolkit_registry
>>> from openff.toolkit.topology.molecule import Molecule
>>> molecule = Molecule.from_smiles('Cc1ccccc1')
>>> smiles_via_openeye = toolkit_registry.call('to_smiles', molecule)
>>> print(smiles_via_openeye)
[H]c1c(c(c(c(c1[H])[H])C([H])([H])[H])[H])[H]

>>> toolkit_registry.deregister_toolkit(OpenEyeToolkitWrapper)
>>> smiles_via_rdkit = toolkit_registry.call('to_smiles', molecule)
>>> print(smiles_via_rdkit)
[H][c]1[c]([H])[c]([H])[c]([C]([H])([H])[H])[c]([H])[c]1[H]
```

12.2 Serialization support

12.3 Collections

Custom collections for the toolkit

12.4 Miscellaneous utilities

Miscellaneous utility functions.

inherit_docstrings	Inherit docstrings from parent class
all_subclasses	Recursively retrieve all subclasses of the specified class
temporary_cd	Context to temporary change the working directory.
get_data_file_path	Get the full path to one of the reference files in test-systems.
convert_0_1_smirnoff_to_0_2	Convert an 0.1-compliant SMIRNOFF dict to an 0.2-compliant one.

continues on next page

Table 4 – continued from previous page

convert_0_2_smirnoff_to_0_3	Convert an 0.2-compliant SMIRNOFF dict to an 0.3-compliant one.
get_molecule_parameterIDs	Process a list of molecules with a specified SMIRNOFF ffxml file and determine which parameters are used by which molecules, returning collated results.
unit_to_string	Serialize a openmm.unit.Unit and return it as a string.

12.4.1 openff.toolkit.utils.utils.inherit_docstrings

openff.toolkit.utils.utils.**inherit_docstrings**(*cls*)
Inherit docstrings from parent class

12.4.2 openff.toolkit.utils.utils.all_subclasses

openff.toolkit.utils.utils.**all_subclasses**(*cls*)
Recursively retrieve all subclasses of the specified class

12.4.3 openff.toolkit.utils.utils.temporary_cd

openff.toolkit.utils.utils.**temporary_cd**(*dir_path*)
Context to temporary change the working directory.

Parameters

dir_path [str] The directory path to enter within the context

Examples

```
>>> dir_path = '/tmp'
>>> with temporary_cd(dir_path):
...     pass  # do something in dir_path
```

12.4.4 openff.toolkit.utils.utils.get_data_file_path

openff.toolkit.utils.utils.**get_data_file_path**(*relative_path*)
Get the full path to one of the reference files in testsystems. In the source distribution, these files are in `openff/toolkit/data/`, but on installation, they're moved to somewhere in the user's python site-packages directory.

Parameters

name [str] Name of the file to load (with respect to the repex folder).

12.4.5 openff.toolkit.utils.utils.convert_0_1_smirnoff_to_0_2

openff.toolkit.utils.utils.**convert_0_1_smirnoff_to_0_2**(*smirnoff_data_0_1*)

Convert an 0.1-compliant SMIRNOFF dict to an 0.2-compliant one. This involves renaming several tags, adding Electrostatics and ToolkitAM1BCC tags, and separating improper torsions into their own section.

Parameters

smirnoff_data_0_1 [dict] Hierarchical dict representing a SMIRNOFF data structure according the the 0.1 spec

Returns

smirnoff_data_0_2 Hierarchical dict representing a SMIRNOFF data structure according the the 0.2 spec

12.4.6 openff.toolkit.utils.utils.convert_0_2_smirnoff_to_0_3

openff.toolkit.utils.utils.**convert_0_2_smirnoff_to_0_3**(*smirnoff_data_0_2*)

Convert an 0.2-compliant SMIRNOFF dict to an 0.3-compliant one. This involves removing units from header tags and adding them to attributes of child elements. It also requires converting ProperTorsions and ImproperTorsions potentials from “charmm” to “fourier”.

Parameters

smirnoff_data_0_2 [dict] Hierarchical dict representing a SMIRNOFF data structure according the the 0.2 spec

Returns

smirnoff_data_0_3 Hierarchical dict representing a SMIRNOFF data structure according the the 0.3 spec

12.4.7 openff.toolkit.utils.utils.get_molecule_parameterIDs

openff.toolkit.utils.utils.**get_molecule_parameterIDs**(*molecules*, *forcefield*)

Process a list of molecules with a specified SMIRNOFF ffxml file and determine which parameters are used by which molecules, returning collated results.

Parameters

molecules [list of openff.toolkit.topology.Molecule] List of molecules (with explicit hydrogens) to parse

forcefield [openff.toolkit.typing.engines.smirnoff.ForceField] The ForceField to apply

Returns

parameters_by_molecule [dict] Parameter IDs used in each molecule, keyed by isomeric SMILES generated from provided OEMols. Each entry in the dict is a list which does not necessarily have unique entries; i.e. parameter IDs which are used more than once will occur multiple times.

parameters_by_ID [dict] Molecules in which each parameter ID occur, keyed by parameter ID. Each entry in the dict is a set of isomeric SMILES for molecules in which that parameter occurs. No frequency information is stored.

12.4.8 openff.toolkit.utils.utils.unit_to_string

openff.toolkit.utils.utils.**unit_to_string**(*input_unit*)

Serialize a openmm.unit.Unit and return it as a string.

Parameters

input_unit [A openmm.unit.Unit] The Unit object to serialize

Returns

unit_string [str] The serialized unit.

INDEX