# CONTENTS

1 Design Principles 3
   1.1 Scope and Features of GMSO .............................................. 3
   1.2 Structure of GMSO .......................................................... 4

2 Data Structures in GMSO 5
   2.1 Core Classes ................................................................. 5
   2.2 Potential Classes ........................................................... 13
   2.3 ForceField ................................................................. 19

3 Formats 23
   3.1 GROMACS ................................................................. 23
   3.2 GSD ........................................................................... 23
   3.3 xyz ......................................................................... 23
   3.4 LAMMPS DATA .............................................................. 23

4 External 25
   4.1 mBuild ................................................................. 25
   4.2 Parmed ................................................................. 25
   4.3 OpenMM ................................................................. 25

5 Installation 27
   5.1 Installing with conda ...................................................... 27
   5.2 Installing from source conda ........................................... 27
   5.3 Install an editable version from source ................................. 27
   5.4 Supported Python Versions ............................................... 28
   5.5 Testing your installation .................................................. 28
   5.6 Install pre-commit .......................................................... 28
   5.7 Building the documentation ............................................. 28

6 Using GMSO with Docker 29
   6.1 Prerequisites ............................................................... 29
   6.2 Quick Start ................................................................. 29
   6.3 Persisting User Volumes .................................................. 30
   6.4 Running Python scripts in the container ................................ 30
   6.5 Cleaning Up ............................................................... 31

7 Contributing 33
   7.1 Features ................................................................. 33
   7.2 Version control ............................................................ 33
   7.3 Source code ............................................................... 33
   7.4 Tests ................................................................. 34
This is the documentation for GMSO, the General Molecular Simulation Object. It is a part of the MoSDeF, the Molecular Simulation Design Framework.
1.1 Scope and Features of GMSO

GMSO is designed to enable the flexible, general representation of chemical topologies for molecular simulation. Efforts are made to enable lossless, bias-free storage of data, without assuming particular chemistries, models, or using any particular engine’s ecosystem as a starting point. The scope is generally restrained to the preparation, manipulation, and conversion of and of input files for molecular simulation, i.e. before engines are called to execute the simulations themselves. GMSO currently does not support conversions between trajectory file formats for analysis codes. In the scope of molecular simulation, we loosely define a chemical topology as everything needed to reproducibly prepare a chemical system for simulation. This includes particle coordinates and connectivity, box information, force field data (functional forms, parameters tagged with units, partial charges, etc.) and some optional information that may not apply to all systems (i.e. specification of elements with each particle).

GMSO enables the following features:

- Supporting a variety of models in the molecular simulation/computational chemistry community: No assumptions are made about an interaction site representing an atom or bead, instead supported atomistic, united-atom/coarse-grained, polarizable, and other models!
- Greater flexibility for exotic potentials: The AtomType (and analogue classes for intramolecular interactions) uses sympy to store any potential that can be represented by a mathematical expression. If you can write it down, it can be stored!
- Easier development for glue to new engines: by not being designed for compatibility with any particular molecular simulation engine or ecosystem, it becomes more tractable for developers in the community to add glue for engines that are not currently supported (and even ones that do not exist at present)!
- Compatibility with existing community tools: No single molecular simulation tool will be a silver bullet, so GMSO includes functions to convert objects. These can be used in their own right to convert between objects in-memory and also to support conversion to file formats not natively supported at any given time. Currently supported conversions include ParmEd, OpenMM, mBuild, MDTraj, with others coming in the future!
- Native support for reading and writing many common file formats (XYZ, GRO, TOP, LAMMPSDATA) and indirect support, through other libraries, for many more!
1.2 Structure of GMSO

There are three main modules within the Python package:

- `gmso.core` stores the classes that constitute the core data structures.
- `gmso.formats` stores readers and writers for (on-disk) file formats.
- `gmso.external` includes functions that convert core data structures between external libraries and their internal representation.
Following data structures are available within GMSO.

## 2.1 Core Classes

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmso.Topology</td>
<td>A topology.</td>
</tr>
<tr>
<td>gmso.Atom</td>
<td>An atom represents a single element association in a topology.</td>
</tr>
<tr>
<td>gmso.Bond</td>
<td>A 2-partner connection between sites.</td>
</tr>
<tr>
<td>gmso.Angle</td>
<td>A 3-partner connection between Atoms.</td>
</tr>
<tr>
<td>gmso.Dihedral</td>
<td>A 4-partner connection between sites.</td>
</tr>
<tr>
<td>gmso.Improper</td>
<td>A 4-partner connection between sites.</td>
</tr>
<tr>
<td>gmso.AtomType</td>
<td>A description of non-bonded interactions between sites.</td>
</tr>
<tr>
<td>gmso.BondType</td>
<td>A description of the interaction between 2 bonded partners.</td>
</tr>
<tr>
<td>gmso.AngleType</td>
<td>A description of the interaction between 3 bonded partners.</td>
</tr>
<tr>
<td>gmso.DihedralType</td>
<td>A description of the interaction between 4 bonded partners.</td>
</tr>
<tr>
<td>gmso.ImproperType</td>
<td>A description of the interaction between 4 bonded partners.</td>
</tr>
</tbody>
</table>

### 2.1.1 Topology

```python
class gmso.Topology(name='Topology', box=None)
```

A topology.

A topology represents a chemical structure wherein lie the collection of sites which together form a chemical structure containing connections (gmso.Bond, gmso.Angle and gmso.Dihedral (along with their associated types). A topology is the fundamental data structure in GMSO, from which we can gather various information about the chemical structure and apply a forcefield before converting the structure into a format familiar to various simulation engines.

#### Parameters

- **name** (*str, optional, default='Topology'*) – A name for the Topology.
- **box** (*gmso.Box, optional, default=None*) – A gmso.Box object bounding the topology
Variables

- **typed**(bool) – True if the topology is typed
- **combining_rule**(str, ['lorentz', 'geometric']) – The combining rule for the topology, can be either 'lorentz' or 'geometric'
- **scaling_factors**(dict) – A collection of scaling factors used in the forcefield
- **n_sites**(int) – Number of sites in the topology
- **n_connections**(int) – Number of connections in the topology (Bonds, Angles, Dihedrals, Impropers)
- **n_bonds**(int) – Number of bonds in the topology
- **n_angles**(int) – Number of angles in the topology
- **n_dihedrals**(int) – Number of dihedrals in the topology
- **n_impropers**(int) – Number of impropers in the topology
- **connections**(tuple of gmso.Connection objects) – A collection of bonds, angles, dihedrals, and impropers in the topology
- **bonds**(tuple of gmso.Bond objects) – A collection of bonds in the topology
- **angles**(tuple of gmso.Angle objects) – A collection of angles in the topology
- **dihedrals**(tuple of gmso.Dihedral objects) – A collection of dihedrals in the topology
- **impropers**(tuple of gmso.Improper objects) – A collection of impropers in the topology
- **connection_types**(tuple of gmso.Potential objects) – A collection of BondTypes, AngleTypes, DihedralTypes, and ImproperTypes in the topology
- **atom_types**(tuple of gmso.AtomType objects) – A collection of AtomTypes in the topology
- **bond_types**(tuple of gmso.BondType objects) – A collection of BondTypes in the topology
- **angle_types**(tuple of gmso.AngleType objects) – A collection of AngleTypes in the topology
- **dihedral_types**(tuple of gmso.DihedralType objects) – A collection of DihedralTypes in the topology
- **improper_types**(tuple of gmso.ImproperType objects) – A collection of ImproperTypes in the topology
- **pairpotential_types**(tuple of gmso.PairPotentialType objects) – A collection of PairPotentialTypes in the topology
- **atom_type_expressions**(list of gmso.AtomType.expression objects) – A collection of all the expressions for the AtomTypes in topology
- **connection_type_expressions**(list of gmso.Potential.expression objects) – A collection of all the expressions for the Potential objects in the topology that represent a connection type
- **bond_type_expressions**(list of gmso.BondType.expression objects) – A collection of all the expressions for the BondTypes in topology
• **angle_type_expressions** *(list of gmso.AngleType.expression objects)* – A collection of all the expressions for the AngleTypes in topology

• **dihedral_type_expressions** *(list of gmso.DihedralType.expression objects)* – A collection of all the expression for the DihedralTypes in the topology

• **improper_type_expressions** *(list of gmso.ImproperType.expression objects)* – A collection of all the expression for the ImproperTypes in the topology

• **pairpotential_type_expressions** *(list of gmso.PairPotentialType.expression objects)* – A collection of all the expression for the PairPotentialTypes in the topology

**add_connection**(*connection, update_types=False*)

Add a gmso.Connection object to the topology.

This method will add a gmso.Connection object to the topology, it can be used to generically include any Connection object i.e. Bond or Angle or Dihedral to the topology. According to the type of object added, the equivalent collection in the topology is updated. For example- If you add a Bond, this method will update topology.connections and topology.bonds object. Additionally, if update_types is True (default behavior), it will also update any Potential objects associated with the connection.

**Parameters**

- **connection** *(one of gmso.Connection, gmso.Bond, gmso.Angle, gmso.Dihedral, or gmso.Improper object)*

- **update_types** *(bool, default=True)* – If True also add any Potential object associated with connection to the topology.

**Returns**

The Connection object or equivalent Connection object that is in the topology

**Return type**

gmso.Connection

**add_site**(*site, update_types=False*)

Add a site to the topology.

This method will add a site to the existing topology, since sites are stored in an indexed set, adding redundant site will have no effect. If the update_types parameter is set to true (default behavior), this method will also check if there is an gmso.AtomType associated with the site and it to the topology’s AtomTypes collection.

**Parameters**

- **site** *(gmso.core.Site)* – Site to be added to this topology

- **update_types** *(bool, default=True)* – If true, add this site’s atom type to the topology’s set of AtomTypes

**update_topology**()

Update the entire topology.
2.1.2 SubTopology

2.1.3 Atom

class gmso.Atom(name="", label="", group=None, molecule=None, residue=None, position=None, charge=None, mass=None, element=None, atom_type=None)

An atom represents a single element association in a topology.
Atoms are the representation of an element within gmso that describes any general atom in a molecular simulation. Atoms also contain information that are unique to elements vs other types of interaction sites in molecular simulations. For example, charge, mass, and periodic table information.

Parameters

- **name** (str, default="") – Name of the atom, defaults to class name
- **label** (str, default="") – Label to be assigned to the atom
- **group** (StrictStr, default=None) – Flexible alternative label relative to atom
- **molecule** (Molecule, default=None) – Molecule label for the atom, format of (molecule_name, molecule_number)
- **residue** (Residue, default=None) – Residue label for the atom, format of (residue_name, residue_number)
- **position** (Union[Sequence[float], numpy.ndarray, unyt.array.unyt_array], default=None) – The 3D Cartesian coordinates of the position of the atom
- **charge** (Union[unyt.array.unyt_quantity, float, NoneType], default=None) – Charge of the atom
- **mass** (Union[unyt.array.unyt_quantity, float, NoneType], default=None) – Mass of the atom
- **element** (<class ‘gmso.core.element.Element’>, default=None) – Element associated with the atom
- **atom_type** (<class ‘gmso.core.atom_type.AtomType’>, default=None) – AtomType associated with the atom

Notes

Atoms have all the attributes inherited from the base Site class. The order of precedence when attaining properties charge and mass is:

1. atom.charge > atom.atom_type.charge
2. atom.mass > atom.atom_type.mass
Examples

```python
>>> from gmso.core.atom import Atom
>>> atom1 = Atom(name='lithium')
```

See also:

gmso.abc.AbstractSite
An Abstract Base class for implementing site objects in GMSO. The class Atom bases from the gmso.abc.abstract site class.

class Config
Pydantic configuration for the atom class.

    property atom_type: AtomType | None
    Return the atom_type associated with the atom.

    property charge: unyt_quantity | None
    Return the charge of the atom.

    clone()
    Clone this atom.

    property element: Element | None
    Return the element associated with the atom.

classmethod is_valid_charge(charge)
Ensure that the charge is physically meaningful.

classmethod is_valid_mass(mass)
Ensure that the mass is physically meaningful.

    property mass: unyt_quantity | None
    Return the mass of the atom.

2.1.4 Bond

class gmso.Bond(name='', connection_members=None, bond_type=None, restraint=None)
A 2-partner connection between sites.

This is a subclass of the gmso.abc.Connection superclass. This class has strictly 2 members in its connection_members. The connection_type in this class corresponds to gmso.BondType.

Parameters

- name (str, default='') – Name of the bond. Defaults to class name
- bond_members (Tuple[gmso.core.atom.Atom, gmso.core.atom.Atom], default=None) – The 2 atoms involved in the bond.
- bond_type (gmso.core.bond_type.BondType, default=None) – BondType of this bond.
- restraint (dict, default=None) – Restraint for this bond, must be a dict with the following keys: ‘b0’ (unit of length), ‘kb’ (unit of energy/(mol * length**2)). Refer to https://manual.gromacs.org/current/reference-manual/topologies/topology-file-formats.html for more information.


Notes

Inherits some methods from Connection:

__eq__, __repr__, _validate methods.

Additional _validate methods are presented.

class Config

Pydantic configuration for Bond.

property bond_type

Return parameters of the potential type.

property connection_type

Return parameters of the potential type.

equivalent_members()

Get a set of the equivalent connection member tuples.

Returns

A unique set of tuples of equivalent connection members

Return type

tuple

Notes

For a bond:

i, j == j, i

where i and j are the connection members.

property restraint

Return the restraint of this bond.

2.1.5 Angle

class gmso.Angle(name=", connection_members=None, angle_type=None, restraint=None)

A 3-partner connection between Atoms.

This is a subclass of the gmso.Connection superclass. This class has strictly 3 members in its connection members. The connection_type in this class corresponds to gmso.AngleType.

Parameters

• name (str, default=”) – Name of the angle. Defaults to class name


• angle_type (<class 'gmso.core.angle_type.AngleType'>, default=None) – Angle-Type of this angle.

• restraint (dict, default=None) – Restraint for this angle, must be a dict with the following keys: ‘k’ (unit of energy/mol), ‘theta_eq’ (unit of angle), ‘n’ (multiplicity, unitless). Refer to https://manual.gromacs.org/current/reference-manual/topologies/topology-file-formats.html for more information.
Notes

Inherits some methods from Connection:
__eq__, __repr__, _validate methods

Additional _validate methods are presented

```python
class Config
    Support pydantic configuration for attributes and behavior.

property angle_type
    Return the angle type if the angle is parametrized.

property connection_type
    Return the angle type if the angle is parametrized.

equivalent_members()
    Return a set of the equivalent connection member tuples.

Returns
    A unique set of tuples of equivalent connection members

Return type
    frozenset
```

Notes

For an angle:
    i, j, k == k, j, i

where i, j and k are the connection members.

```python
property restraint
    Return the restraint of this angle.
```

2.1.6 Dihedral

```python
class gmso.Dihedral(name="", connection_members=None, dihedral_type=None, restraint=None)
```

A 4-partner connection between sites.

This is a subclass of the gmso.Connection superclass. This class has strictly 4 members in its connection_members. The connection_type in this class corresponds to gmso.DihedralType. The connectivity of a dihedral is:

m1–m2–m3–m4

where m1, m2, m3, and m4 are connection members 1-4, respectively.

Parameters

- name (str, default="") – Name of the dihedral. Defaults to class name
- dihedral_type (class `gmso.core.dihedral_type.DihedralType`, default=None) – DihedralType of this dihedral.
• **restraint** (*dict, default=None*) – Restraint for this dihedral, must be a dict with the following keys: ‘k’ (unit of energy/(mol * angle**2)), ‘phi_eq’ (unit of angle), ‘delta_phi’ (unit of angle). Refer to https://manual.gromacs.org/current/reference-manual/topologies/topology-file-formats.html for more information.

Notes

Inherits some methods from Connection:

- **__eq__, __repr__, _validate** methods

Additional _validate methods are presented

```python
class Config
equivalent_members()
    Get a set of the equivalent connection member tuples

    Returns
    A unique set of tuples of equivalent connection members

    Return type
    frozenset
```

Notes

For a dihedral:

i, j, k, l == l, k, j, i

where i, j, k, and l are the connection members.

```python
property restraint
    Return the restraint of this dihedral.
```

### 2.1.7 Improper

```python
class gmso.Improper(*name="", connection_members=None, improper_type=None*)
```

A 4-partner connection between sites.

This is a subclass of the gmso.Connection superclass. This class has strictly 4 members in its connection_members. The connection_type in this class corresponds to gmso.ImproperType. The connectivity of an improper is:

```
m2 | m1 / m3 m4
```

where m1, m2, m3, and m4 are connection members 1-4, respectively.

Parameters

- **name** (*str, default="") – Name of the improper. Defaults to class name
- **improper_members** (*Tuple[gmso.core.atom.Atom, gmso.core.atom.Atom, gmso.core.atom.Atom, gmso.core.atom.Atom], default=None*) – The 4 atoms of this improper. Central atom first, then the three atoms connected to the central site.
- **improper_type** (*<class 'gmso.core.improper_type.ImproperType'>, default=None*) – ImproperType of this improper.

Notes
Inherits some methods from Connection: 
  __eq__, __repr__, _validate methods

Additional _validate methods are presented

class Config
    Pydantic configuration to link fields to their public attribute.

property connection_type
    Return Potential object for this connection if it exists.

equivalent_members()
    Get a set of the equivalent connection member tuples.
    Returns
        A unique set of tuples of equivalent connection members
    Return type
        frozenset

Notes
    For an improper:
        i, j, k, l == i, k, j, l
    where i, j, k, and l are the connection members.

property improper_type
    Return Potential object for this connection if it exists.

2.2 Potential Classes

2.2.1 AtomType

class gmso.AtomType(name='AtomType', mass=unyt_quantity(0., 'g/mol'), charge=unyt_quantity(0., 'C'), expression=None, parameters=None, potential_expression=None, independent_variables=None, atomclass='', doi='', overrides=None, definition='', description='', tags=None)

A description of non-bonded interactions between sites.

Parameters

  • name (str, default=”) – The name of the potential. Defaults to class name
  • potential_expression (PotentialExpression, default=<PotentialExpression, expression: a*x + b, 1 independent variables>) – The mathematical expression for the potential
  • tags (Any, default={}) – Tags associated with the potential
  • mass (unyt_array, default=0.0 g/mol) – The mass of the atom type
  • charge (unyt_array, default=0.0 C) – The charge of the atom type
  • atomclass (str, default=”) – The class of the atomtype
  • doi (str, default=”) – Digital Object Identifier of publication where this atom type was introduced
• **overrides** *(str, default=set())* – Set of other atom types that this atom type overrides

• **definition** *(str, default=”)* – SMARTS string defining this atom type

• **description** *(str, default=”)* – Description for the AtomType

• **expression** *(Optional[Union[str, sympy.Expr]], default=None)* – The mathematical expression of the functional form of the potential.

• **parameters** *(Optional[dict], default=None)* – The parameters of the Potential expression and their corresponding values, as unyt quantities

• **independent_variables** *(Optional[Union[set, str]], default=None)* – The independent variables in the Potential’s expression.

### class Config

Pydantic configuration of the attributes for an atom_type.

- **property atomclass**
  - Return the atomclass of the atom_type.

- **property charge**
  - Return the charge of the atom_type.

- **clone**(fast_copy=False)
  - Clone this AtomType, faster alternative to deepcopying.

- **property definition**
  - Return the SMARTS string of the atom_type.

- **property description**
  - Return the description of the atom_type.

- **property doi**
  - Return the doi of the atom_type.

- **property mass**
  - Return the mass of the atom_type.

- **property overrides**
  - Return the overrides of the atom_type.

- **classmethod validate_charge**(charge)
  - Check to see that a charge is a unyt array of the right dimension.

- **classmethod validate_mass**(mass)
  - Check to see that a mass is a unyt array of the right dimension.

### 2.2.2 BondType

- **class gmso.BondType**(name='BondType', expression=None, parameters=None,
  independent_variables=None, potential_expression=None,
  member_types=None, member_classes=None, tags=None)

A description of the interaction between 2 bonded partners.

This is a subclass of the gmso.core.Potential superclass. BondType represents a bond type and includes the functional form describing its interactions. The functional form of the potential is stored as a sympy expression and the parameters, with units, are stored explicitly. The AtomTypes that are used to define the bond type are stored as member_types.
Parameters

- **name** *(str, default=*) – The name of the potential. Defaults to class name
- **potential_expression** *(PotentialExpression, default=<PotentialExpression, expression: a*x + b, 1 independent variables>) – The mathematical expression for the potential
- **tags** *(Any, default={}) – Tags associated with the potential
- **member_types** *(Tuple[str, str], default=None) – List-like of gmso.AtomType.name defining the members of this bond type
- **member_classes** *(Tuple[str, str], default=None) – List-like of gmso.AtomType.atomclass defining the members of this bond type
- **expression** *(Optional[Union[str, sympy.Expr]], default=None) – The mathematical expression of the functional form of the potential.
- **parameters** *(Optional[dict], default=None) – The parameters of the Potential expression and their corresponding values, as unyt quantities
- **independent_variables** *(Optional[Union[set, str]], default=None) – The independent variables in the Potential’s expression.

Notes

Inherits many functions from gmso.ParametricPotential: 
__eq__, __validate functions

```python
class Config
    Pydantic configuration for class attributes.

property member_types
    Return the members involved in this bondtype.
```

### 2.2.3 AngleType

```python
class gmso.AngleType(name='AngleType', expression=None, parameters=None,
                        independent_variables=None, potential_expression=None,
                        member_types=None, member_classes=None, tags=None)
```

A description of the interaction between 3 bonded partners.

This is a subclass of the gmso.core.Potential superclass. AngleType represents an angle type and includes the functional form describing its interactions. The functional form of the potential is stored as a `sympy` expression and the parameters, with units, are stored explicitly. The AtomTypes that are used to define the angle type are stored as `member_types`.

Parameters

- **name** *(str, default=*) – The name of the potential. Defaults to class name
- **potential_expression** *(PotentialExpression, default=<PotentialExpression, expression: a*x + b, 1 independent variables>) – The mathematical expression for the potential
- **tags** *(Any, default={}) – Tags associated with the potential
• **member_types** (*Tuple[str, str, str], default=None*) – List-like of `gmso.AtomType.name` defining the members of this angle type

• **member_classes** (*Tuple[str, str, str], default=None*) – List-like of `gmso.AtomType.atomclass` defining the members of this angle type

• **expression** (*Optional[Union[str, sympy.Expr]], default=None*) – The mathematical expression of the functional form of the potential.

• **parameters** (*Optional[dict], default=None*) – The parameters of the Potential expression and their corresponding values, as `unyt` quantities

• **independent_variables** (*Optional[Union[set, str]], default=None*) – The independent variables in the Potential’s expression.

### Notes

Inherits many functions from `gmso.ParametricPotential`:

- `__eq__`, `__validate` functions

```python
class Config

2.2.4 DihedralType

class gmso.DihedralType(name='DihedralType', expression=None, parameters=None, independent_variables=None, potential_expression=None, member_types=None, member_classes=None, tags=None)
```

A description of the interaction between 4 bonded partners.

This is a subclass of the `gmso.core.Potential` superclass. DihedralType represents a dihedral type and includes the functional form describing its interactions. The functional form of the potential is stored as a `sympy` expression and the parameters, with units, are stored explicitly. The `AtomTypes` that are used to define the dihedral type are stored as `member_types`. The connectivity of a dihedral is:

\[
m_1–m_2–m_3–m_4
\]

where \(m_1\), \(m_2\), \(m_3\), and \(m_4\) are connection members 1-4, respectively.

### Parameters

- **name** (*str, default=“") – The name of the potential. Defaults to class name

- **potential_expression** (*PotentialExpression, default=<PotentialExpression, expression: a*x + b, 1 independent variables>) – The mathematical expression for the potential

- **tags** (*Any, default={}) – Tags associated with the potential

- **member_types** (*Tuple[str, str, str, str], default=None*) – List-like of `gmso.AtomType.name` defining the members of this dihedral type

- **member_classes** (*Tuple[str, str, str, str], default=None*) – List-like of `gmso.AtomType.atomclass` defining the members of this dihedral type

- **expression** (*Optional[Union[str, sympy.Expr]], default=None*) – The mathematical expression of the functional form of the potential.

- **parameters** (*Optional[dict], default=None*) – The parameters of the Potential expression and their corresponding values, as `unyt` quantities
• **independent_variables** *(Optional[Union[set, str]], default=None)* – The independent variables in the Potential’s expression.

**Notes**

Inherits many functions from gmso.ParametricPotential:

---

```
class Config
```

---

### 2.2.5 ImproperType

```python
class gmso.ImproperType(name='ImproperType', expression=None, parameters=None, independent_variables=None, potential_expression=None, member_types=None, member_classes=None, tags=None)
```

A description of the interaction between 4 bonded partners.

This is a subclass of the gmso.core.Potential superclass. ImproperType represents a improper type and includes the functional form describing its interactions. The functional form of the potential is stored as a `sympy` expression and the parameters, with units, are stored explicitly. The AtomTypes that are used to define the improper type are stored as *member_types*. The connectivity of an improper is:

\[
m_2 \mid m_1 / m_3 m_4
\]

where m1, m2, m3, and m4 are connection members 1-4, respectively.

**Parameters**

- **name** *(str, default=’’)* – The name of the potential. Defaults to class name
- **potential_expression** *(PotentialExpression, default=<PotentialExpression, expression: a*x + b, 1 independent variables>)* – The mathematical expression for the potential
- **tags** *(Any, default={})* – Tags associated with the potential
- **member_types** *(Tuple[str, str, str, str], default=None)* – List-like of gmso.AtomType.name defining the members of this improper type
- **member_classes** *(Tuple[str, str, str, str], default=None)* – List-like of gmso.AtomType.atomclass defining the members of this improper type
- **expression** *(Optional[Union[str, sympy.Expr]], default=None)* – The mathematical expression of the functional form of the potential.
- **parameters** *(Optional[dict], default=None)* – The parameters of the Potential expression and their corresponding values, as *unyt* quantities
- **independent_variables** *(Optional[Union[set, str]], default=None)* – The independent variables in the Potential’s expression.

**Notes**

Inherits many functions from gmso.ParametricPotential:

---

```
__eq__, _validate functions
```

---

2.2. Potential Classes

---

17
class Config
    Pydantic configuration for attributes.

property member_types
    Return member information for this ImproperType.

2.2.6 PairPotentialType

class gmso.PairPotentialType (name='PairPotentialType', expression=None, parameters=None,
                           independent_variables=None, potential_expression=None,
                           member_types=None, tags=None)

A description of custom pairwise potential between 2 AtomTypes that does not follow combination
rule.

This is a subclass of the gmso.core.ParametricPotential superclass. PairPotentialType represents a
type of pairwise potential between two Atomtypes that does not follow a specific combination rule,
and includes the functional form describing its interactions. The functional form of the potential is
stored as a sympy expression and the parameters, with units, are stored explicitly. The AtomTypes
that are used to define the dihedral type are stored as member_types.

Parameters

• name (str, default="") – The name of the potential. Defaults to class name

• potential_expression (PotentialExpression, default=<PotentialExpression, expression: a*x + b, 1 independent variables>) – The mathematical expression for the potential

• tags (Any, default=[]) – Tags associated with the potential

• member_types (Tuple[str, str], default=None) – List-like of strs, referring to
gmso.Atomtype.name or gmso.Atomtype.atomclass, defining the members of this
pair potential type

• expression (Optional[Union[str, sympy.Expr]], default=None) – The mathematical
expression of the functional form of the potential.

• parameters (Optional[dict], default=None) – The parameters of the Potential ex-
pression and their corresponding values, as unyt quantities

• independent_variables (Optional[Union[set, str]], default=None) – The indepen-
dent variables in the Potential’s expression.

Notes

Inherits many functions from gmso.ParametricPotential:

    __eq__, __validate functions


class Config
2.3 ForceField

```python
class gmso.ForceField(xml_loc=None, strict=True, greedy=True, backend='forcefield-utilities')
```

A generic implementation of the forcefield class.

The ForceField class is one of the core data structures in gmso, which is used to hold a collection of gmso.core.Potential subclass objects along with some metadata to represent a forcefield. The forcefield object can be applied to any gmso.Topology which has effects on its Sites, Bonds, Angles and Dihedrals.

**Parameters**

- `xml_loc (str)` – Path to the forcefield xml. The forcefield xml can be either in Foyer or GM SO style.
- `strict (bool, default=True)` – If true, perform a strict validation of the forcefield XML file
- `greedy (bool, default=True)` – If True, when using strict mode, fail on the first error/mismatch
- `backend (str, default="gmso")` – Can be “gmso” or “forcefield-utilities”. This will define the methods to load the forcefield.

**Variables**

- `name (str)` – Name of the forcefield
- `version (str)` – Version of the forcefield
- `atom_types (dict)` – A collection of atom types in the forcefield
- `bond_types (dict)` – A collection of bond types in the forcefield
- `angle_types (dict)` – A collection of angle types in the forcefield
- `dihedral_types (dict)` – A collection of dihedral types in the forcefield
- `units (dict)` – A collection of unyt.Unit objects used in the forcefield
- `scaling_factors (dict)` – A collection of scaling factors used in the forcefield

See also:

- `gmso.ForceField.from_xml` A class method to create forcefield object from XML files
- `gmso.utils.ff_utils.validate` Function to validate the gmso XML file

**property atom_class_groups**

Return a dictionary of atomClasses in the Forcefield.

**classmethod from_xml(xmls_or_etrees, strict=True, greedy=True)**

Create a gmso.Forcefield object from XML File(s).

This class method creates a ForceField object from the reference XML file. This method takes in a single or collection of XML files with information about gmso.AtomTypes, gmso.BondTypes, gmso.AngleTypes, gmso.PairPotentialTypes and gmso.DihedralTypes to create the ForceField object.

**Parameters**
• **xmls_or_etrees** *(Union[str, Iterable[str], etree._ElementTree, Iterable[etree._ElementTree]])* – The forcefield XML locations or XML Element Trees
• **strict** *(bool, default=True)* – If true, perform a strict validation of the forcefield XML file
• **greedy** *(bool, default=True)* – If True, when using strict mode, fail on the first error/mismatch

Returns

**forcefield** – A gmso.Forcefield object with a collection of Potential objects created using the information in the XML file

**Return type**

gmso.ForceField

---

**get_parameters** *(group, key, warn=False, copy=False)*

Return parameters for a specific potential by key in this ForceField.

This function uses the `get_potential` function to get Parameters

See also:

**gmso.ForceField.get_potential**

Get specific potential/parameters from a forcefield potential group by key

**get_potential** *(group, key, return_match_order=False, warn=False)*

Return a specific potential by key in this ForceField.

**Parameters**

• **group** *( {'atom_type', 'bond_type', 'angle_type', 'dihedral_type', 'improper_type' } )* – The potential group to perform this search on
• **key** *(str (for atom type) or list of str (for connection types))* – The key to lookup for this potential group
• **return_match_order** *(bool, default=False)* – If true, return the order of connection member types/classes that got matched
• **warn** *(bool, default=False)* – If true, raise a warning instead of Error if no match found

**Returns**

The parametric potential requested

**Return type**

gmso.ParametricPotential

**Raises**

**MissingPotentialError** – When the potential specified by `key` is not found in the ForceField potential group `group`

---

**group_angle_types_by_expression()**

Return all AngleTypes in this ForceField with grouped by expression.

See also:

**_group_by_expression**

Groups a dictionary of gmso.ParametricPotentials by their expression

**Returns**

A dictionary where the key, value -> expression, list of AngleTypes with that expression

**Return type**

dict

**group_atom_types_by_expression()**

Return all AtomTypes in this ForceField with grouped by expression.
See also:

_group_by_expression

Groups a dictionary of gmso.ParametricPotentials by their expression

Returns

A dictionary where the key, value -> expression, list of atom_types with that expression

Return type
dict

group_bond_types_by_expression()

Return all BondTypes in this ForceField with grouped by expression.

See also:

_group_by_expression

Groups a dictionary of gmso.ParametricPotentials by their expression

Returns

A dictionary where the key, value -> expression, list of BondTypes with that expression

Return type
dict

group_dihedral_types_by_expression()

Return all DihedralTypes in this ForceField with grouped by expression.

See also:

_group_by_expression

Groups a dictionary of gmso.ParametricPotentials by their expression

Returns

A dictionary where the key, value -> expression, list of DihedralTypes with that expression

Return type
dict

group_improper_types_by_expression()

Return all ImproperTypes in this ForceField with grouped by expression.

See also:

_group_by_expression

Groups a dictionary of gmso.ParametricPotentials by their expression

Returns

A dictionary where the key, value -> expression, list of ImproperTypes with that expression

Return type
dict

group_pairpotential_types_by_expression()

Return all PairPotentialTypes in this ForceField with grouped by expression

See also:

_group_by_expression

Groups a dictionary of gmso.ParametricPotentials by their expression
Returns
A dictionary where the key, value -> expression, list of PairPotentialTypes with that expression

Return type
dict

property non_element_types
Get the non-element types in the ForceField.

to_xml(filename, overwrite=False, backend='gmso')
Get an lxml ElementTree representation of this ForceField

Parameters
• filename (Union[str, pathlib.Path], default=None) – The filename to write the XML file to
• overwrite (bool, default=False) – If True, overwrite an existing file if it exists
• backend (str, default="gmso") – Can be “gmso” or “forcefield-utilities”. This will define the methods to write the xml.
This submodule provides readers and writers for (on-disk) file formats.

### 3.1 GROMACS

The following methods are available for reading and writing GROMACS files.

### 3.2 GSD

The following methods are available for reading and writing GSD files.

### 3.3 xyz

The following methods are available for reading and writing xyz files.

### 3.4 LAMMPS DATA

The following methods are available for reading and writing LAMMPS data.
This submodule includes functions that convert core data structures between external libraries and their internal representation.

4.1 mBuild

The following methods are available for converting mBuild objects to and from GMSO.

4.2 Parmed

Conversion methods for Parmed objects to and from GMSO.

4.3 OpenMM

Conversion methods for OpenMM objects to and from GMSO.
Chapter Five

Installation

5.1 Installing with conda

Starting from GMsO version 0.3.0, you can use conda to install GMsO in your preferred environment. This will also install the dependencies of GMsO.

```
(your-env) $ conda install -c conda-forge gmso
```

5.2 Installing from source conda

Dependencies of GMsO are listed in the files environment.yml (lightweight environment specification containing minimal dependencies) and environment-dev.yml (comprehensive environment specification including optional and testing packages for developers). The gmso or gmso-dev conda environments can be created with

```
$ git clone https://github.com/mosdef-hub/gmso.git
$ cd gmso
# for gmso conda environment
$ conda env create -f environment.yml
$ conda activate gmso

# for gmso-dev
$ conda env create -f environment-dev.yml
$ conda activate gmso

# install a non-editable version of gmso
$ pip install .
```

5.3 Install an editable version from source

Once all dependencies have been installed and the conda environment has been created, the GMsO itself can be installed.

```
$ cd gmso
$ conda activate gmso-dev # or gmso depending on your installation
$ pip install -e .
```
5.4 Supported Python Versions

Python 3.8-3.11 is the recommend version for users. It is the only version on which development and testing consistently takes place. Older (3.6-3.7) and newer (3.12+) versions of Python 3 are likely to work but no guarantee is made and, in addition, some dependencies may not be available for other versions. No effort is made to support Python 2 because it is considered obsolete as of early 2020.

5.5 Testing your installation

GMSO uses pytest to execute its unit tests. To run them, first install the gmso-dev environment from above as well as gmso itself

```
$ conda activate gmso-dev
$ pip install -e .
```

And then run the tests with the pytest executable:

```
$ pytest -v
```

5.6 Install pre-commit

We use [pre-commit](https://pre-commit.com/) to automatically handle our code formatting and this package is included in the dev environment. With the gmso-dev conda environment active, pre-commit can be installed locally as a git hook by running

```
$ pre-commit install
```

And (optional) all files can be checked by running

```
$ pre-commit run --all-files
```

5.7 Building the documentation

GMSO uses sphinx to build its documentation. To build the docs locally, run the following while in the docs directory:

```
$ conda env create -f docs-env.yml
$ conda activate gmso-docs
$ make html
```
As much of scientific software development happens in unix platforms, to avoid the quirks of development dependent on system you use, a recommended way is to use docker or other containerization technologies. This section is a how to guide on using GMSO with docker.

### 6.1 Prerequisites

A docker installation in your machine. Follow this link to get a docker installation working on your machine. If you are not familiar with docker and want to get started with docker, the Internet is full of good tutorials like the ones here and here.

### 6.2 Quick Start

After you have a working docker installation, please use the following command to use run a jupyter-notebook with all the dependencies for GMSO installed:

```
$ docker pull mosdef/gmso:latest
$ docker run -it --name gmso -p 8888:8888 mosdef/gmso:latest
```

If no command is provided to the container (as above), the container starts a jupyter-notebook at the (container) location `/home/anaconda/data`. To access the notebook, paste the notebook URL into a web browser on your computer. When you are finished, you can control-C to exit the notebook as usual. The docker container will exit upon notebook shutdown.

Alternatively, you can also start a Bourne shell to use python from the container’s terminal:

```
$ docker run -it --mount type=bind,source=$(pwd),target="/home/anaconda/data" mosdef/GMSONOTEBOOK sh
~ $ source .profile
(gmso-dev) ~ $
```

**Warning:** Containers by nature are ephemeral, so filesystem changes (e.g., adding a new notebook) only persist until the end of the container’s lifecycle. If the container is removed, any changes or code additions will not persist. See the section below for persistent data.

Note
The -it flags connect your keyboard to the terminal running in the container. You may run the prior command without those flags, but be aware that the container will not respond to any keyboard input. In that case, you would need to use the docker ps and docker kill commands to shut down the container.

6.3 Persisting User Volumes

If you will be using GMSO from a docker container, a recommended way is to mount what are called user volumes in the container. User volumes will provide a way to persist all filesystem/code additions made to a container regardless of the container lifecycle. For example, you might want to create a directory called gms-o-notebooks in your local system, which will store all your GMSO notebooks/code. In order to make that accessible to the container(where the notebooks will be created/edited), use the following steps:

```bash
$ mkdir -p /path/to/gmso-notebooks
$ cd /path/to/gmso-notebooks
$ docker run -it --name gmso --mount type=bind,source=$(pwd),target=/home/anaconda/data -p 8888:8888 mosdef/gmso:latest
```

You can easily mount a different directory from your local machine by changing `source=$(pwd)` to `source=/path/to/my/favorite/directory`.

**Note:** The --mount flag mounts a volume into the docker container. Here we use a bind mount to bind the current directory on our local filesystem to the /home/anaconda/data location in the container. The files you see in the jupyter-notebook browser window are those that exist on your local machine.

**Warning:** If you are using the container with jupyter notebooks you should use the /home/anaconda/data location as the mount point inside the container; this is the default notebook directory.

6.4 Running Python scripts in the container

Jupyter notebooks are a great way to explore new software and prototype code. However, when it comes time for production science, it is often better to work with python scripts. In order to execute a python script (example.py) that exists in the current working directory of your local machine, run:

```bash
$ docker run --mount type=bind,source=$(pwd),target=/home/anaconda/data mosdef/gmso:latest "python data/test.py"
```

Note that once again we are bind mounting the current working directory to /home/anaconda/data. The command we pass to the container is python data/test.py. Note the prefix data/ to the script; this is because we enter the container in the home folder (/home/anaconda), but our script is located under /home/anaconda/data.

**Warning:** Do not bind mount to target=/home/anaconda. This will cause errors.

If you don't require a Jupyter notebook, but just want a Python interpreter, you can run:

```bash
$ docker run --mount type=bind,source=$(pwd),target=/home/anaconda/data mosdef/gmso:latest python
```
If you don’t need access to any local data, you can of course drop the `--mount` command:

```
$ docker run mosdef/gmso:latest python
```

### 6.5 Cleaning Up

You can remove the created container by using the following command:

```
$ docker container rm gmso
```

**Note:** Instead of using `latest`, you can use the image `mosdef/gmso:stable` for most recent stable release of GMSO and run the tutorials.
Contributions are welcomed via pull requests on GitHub. Developers and/or users will review requested changes and make comments. The rest of this file will serve as a set of general guidelines for contributors.

7.1 Features

7.1.1 Implement functionality in a general and flexible fashion

GMSO is designed to be general and flexible, not limited to single chemistries, file formats, simulation engines, or simulation methods. Additions to core features should attempt to provide something that is applicable to a variety of use-cases and not targeted at only the focus area of your research. However, some specific features targeted toward a limited use case may be appropriate. Speak to the developers before writing your code and they will help you make design choices that allow flexibility.

7.2 Version control

We currently use the “standard” Pull Request model. Contributions should be implemented on feature branches of forks. Please try to keep the master branch of your fork up-to-date with the master branch of the main repository.

7.2.1 Propose a single set of related changes

Small changes are preferred over large changes. A major contribution can often be broken down into smaller PRs. Large PRs that affect many parts of the codebase can be harder to review and are more likely to cause merge conflicts.

7.3 Source code

7.3.1 Use a consistent style

It is important to have a consistent style throughout the source code. The following criteria are desired:

- Lines wrapped to 80 characters
- Lines are indented with spaces
- Lines do not end with whitespace
- For other details, refer to PEP8
To help with the above, there are tools such as flake8 and Black.

### 7.3.2 Document code with comments

All public-facing functions should have docstrings using the numpy style. This includes concise paragraph-style description of what the class or function does, relevant limitations and known issues, and descriptions of arguments. Internal functions can have simple one-liner docstrings.

### 7.4 Tests

#### 7.4.1 Write unit tests

All new functionality in GMSO should be tested with automatic unit tests that execute in a few seconds. These tests should attempt to cover all options that the user can select. All or most of the added lines of source code should be covered by unit test(s). We currently use pytest, which can be executed simply by calling pytest from the root directory of the package.
INDEX

A
add_connection() (gmso.Topology method), 7
add_site() (gmso.Topology method), 7
Angle (class in gmso), 10
Angle.Config (class in gmso), 11
angle_type (gmso.Angle property), 11
AngleType (class in gmso), 15
AngleType.Config (class in gmso), 16
Atom (class in gmso), 8
Atom.Config (class in gmso), 9
atom_class_groups (gmso.ForceField property), 19
atom_type (gmso.Atom property), 9
atomclass (gmso.AtomType property), 14
AtomType (class in gmso), 13
AtomType.Config (class in gmso), 14

B
Bond (class in gmso), 9
Bond.Config (class in gmso), 10
bond_type (gmso.Bond property), 10
BondType (class in gmso), 14
BondType.Config (class in gmso), 15

C
charge (gmso.Atom property), 9
charge (gmso.AtomType property), 14
clone() (gmso.Atom method), 9
clone() (gmso.AtomType method), 14
connection_type (gmso.Angle property), 11
connection_type (gmso.Bond property), 10
connection_type (gmso.Improper property), 13

D
definition (gmso.AtomType property), 14
description (gmso.AtomType property), 14
Dihedral (class in gmso), 11
Dihedral.Config (class in gmso), 12
DihedralType (class in gmso), 16
DihedralType.Config (class in gmso), 17
doi (gmso.AtomType property), 14

e
element (gmso.Atom property), 9
equivalent_members() (gmso.Angle method), 11
equivalent_members() (gmso.Bond method), 10
equivalent_members() (gmso.Dihedral method), 12
equivalent_members() (gmso.Improper method), 13

F
ForceField (class in gmso), 19
from_xml() (gmso.ForceField class method), 19

G
get_parameters() (gmso.ForceField method), 20
gt_potential() (gmso.ForceField method), 20
group_angle_types_by_expression() (gmso.ForceField method), 20
group_atom_types_by_expression() (gmso.ForceField method), 20
group_bond_types_by_expression() (gmso.ForceField method), 21
group_dihedral_types_by_expression() (gmso.ForceField method), 21
group_improper_types_by_expression() (gmso.ForceField method), 21
group_pairpotential_types_by_expression() (gmso.ForceField method), 21

I
Improper (class in gmso), 12
Improper.Config (class in gmso), 13
improper_type (gmso.Improper property), 13
ImproperType (class in gmso), 17
ImproperType.Config (class in gmso), 17
is_valid_charge() (gmso.Atom class method), 9
is_valid_mass() (gmso.Atom class method), 9

M
mass (gmso.Atom property), 9
mass (gmso.AtomType property), 14
member_types (gmso.BondType property), 15
member_types (gmso.ImproperType property), 18
N

non_element_types (gmso.ForceField property), 22

O

overrides (gmso.AtomType property), 14

P

PairPotentialType (class in gmso), 18
PairPotentialType.Config (class in gmso), 18

R

restraint (gmso.Angle property), 11
restraint (gmso.Bond property), 10
restraint (gmso.Dihedral property), 12

T

to_xml() (gmso.ForceField method), 22
Topology (class in gmso), 5

U

update_topology() (gmso.Topology method), 7

V

validate_charge() (gmso.AtomType class method), 14
validate_mass() (gmso.AtomType class method), 14