Introduction to Sire

Christopher Woods

What is Sire?

- Sire is a molecular simulation framework
- Collection of building blocks written in C++
- Use Python to connect the blocks together
- By connecting together different blocks, you can write analysis scripts, glue scripts and even complete simulation programs

Why Sire?

- My research focusses on method development
- Particularly interested in creating new Monte Carlo and free energy algorithms
- Design of Sire is well-suited to developing new Monte Carlo and free energy algorithms
- Also well-suited to studying biomolecules, e.g. complete implementation of the Amber, OPLS and CHARMM forcefields

A simple Sire script: water_dimer.py

```
# Import the Sire Molecule library. This library contains
# everything needed to represent and manipulate atoms
# and molecules
from Sire.Mol import *

# Import the Sire Input/Output library. This library contains
# everything needed to load and save things to and from files
from Sire.IO import *

# Import the Sire Maths library. This library contains mathematical
# objects, e.g. vectors, angles, planes etc.
from Sire.Maths import *

# Use the PDB object from Sire.IO to load a PDB file
# containing a water dimer
water_dimer = PDB().read("input/water_dimer.pdb")

# water_dimer is a container containing the two
# water molecules in the dimer. Lets get hold of
# each water molecule
first_water = water_dimer[MolIdx(0)].molecule()
print("The first water molecule is %s" % first_water)

# Now the second water molecule
second_water = water_dimer[MolIdx(1)].molecule()
print("The second water molecule is %s" % second_water)

# Let's print out all of the data about all of the atoms
# of the first water
print("\nFirst water molecule:")
for i in range(0, first_water.nAtoms()):
    atom = first_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's do the same to the second water
print "\nSecond water molecule:"
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
    print(“%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's calculate the distance between the center of masses
# of the two water molecules
com_first_water = first_water.evaluate().centerOfMass()
com_second_water = second_water.evaluate().centerOfMass()

distance = Vector.distance(com_first_water, com_second_water)
print("\nThe distance between the centers of mass of the waters is %s A" % distance)
```

```
# Import the Sire Molecule library. This library contains
# everything needed to represent and manipulate atoms
# and molecules
from Sire.Mol import *

# Import the Sire Maths library. This library contains mathematical
# objects, e.g. vectors, angles, planes
from Sire.Maths import *

# Use the PDB object from Sire.IO to load a molecule
# containing a water dimer
water_dimer = PDB().read("input/water_dimer.pdb")

# water_dimer is a container containing the two
# water molecules in the dimer. Lets get hold of
# each water molecule
first_water = water_dimer[MolIdx(0)].molecule()
print("The first water molecule is %s" % first_water)

# Now the second water molecule
second_water = water_dimer[MolIdx(1)].molecule()
print("The second water molecule is %s" % second_water)

# Let's print out all of the data about all of the atoms
# of the first water
print("\nFirst water molecule:")
for i in range(0, first_water.nAtoms()):
    atom = first_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's do the same to the second water
print "\nSecond water molecule:"
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's calculate the distance between the center of masses
# of the two water molecules
com_first_water = first_water.evaluate().centerOfMass()
com_second_water = second_water.evaluate().centerOfMass()

distance = Vector.distance(com_first_water, com_second_water)
print("\nThe distance between the centers of mass of the waters is %s A" % distance)
```

Sire is divided into lots of libraries. You import the library you want to use using

from Sire.??? import *

or

import Sire.???

```
# Import the Sire Molecule library. This library contains
# everything needed to represent and manipulate atoms
# and molecules
from Sire.Mol import *

# Import the Sire Input/Output library. This library contains
# everything needed to load and save things to and from files
from Sire.IO import *

# Import the Sire Maths library. This library contains mathematical
# objects, e.g. vectors, angles, planes etc.
from Sire.Maths import *

# Use the PDB object from Sire.IO to load a PDB file
# containing a water dimer
water_dimer = PDB().read("input/water_dimer.pdb")

# water_dimer is a container containing the two
# water molecules in the dimer. Lets get hold of
# each water molecule
first_water = water_dimer[MolIdx(0)].molecule()
print("The first water molecule is %s" % first_water)

# Now the second water molecule
second_water = water_dimer[MolIdx(1)].molecule()
print("The second water molecule is %s" % second_water)

# Let's print out all of the data about all of the atoms
# of the first water
print("\nFirst water molecule:")
for i in range(0, first_water.nAtoms()):
    atom = first_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's do the same to the second water
print "\nSecond water molecule:"
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's calculate the distance between the center of masses
# of the two water molecules
com_first_water = first_water.evaluate().centerOfMass()
com_second_water = second_water.evaluate().centerOfMass()

distance = Vector.distance(com_first_water, com_second_water)
print("\nThe distance between the centers of mass of the waters is %s A" % distance)
```

Sire is divided into lots of libraries. You import the library you want to use using

from Sire.??? import *

or

import Sire.???

```
# Import the Sire Molecule library. This library contains
# everything needed to represent and manipulate atoms
# and molecules
from Sire.Mol import *

# Import the Sire Input/Output library. This library contains
# everything needed to load and save things to and from files
from Sire.IO import *

# Import the Sire Maths library. This library contains mathematical
# objects, e.g. vectors, angles, planes etc.
from Sire.Maths import *

# Use the PDB object from Sire.IO to load a PDB file
# containing a water dimer
water_dimer = PDB().read("input/water_dimer.pdb")

# water_dimer is a container containing the two
# water molecules in the dimer. Lets get hold of
# each water molecule
first_water = water_dimer[MolIdx(0)].molecule()
print("The first water molecule is %s" % first_water)

# Now the second water molecule
second_water = water_dimer[MolIdx(1)].molecule()
print("The second water molecule is %s" % second_water)

# Let's print out all of the data about all of the atoms
# of the first water
print("\nFirst water molecule:")
for i in range(0, first_water.nAtoms()):
    atom = first_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's do the same to the second water
print "\nSecond water molecule:"
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's calculate the distance between the center of masses
# of the two water molecules
com_first_water = first_water.evaluate().centerOfMass()
com_second_water = second_water.evaluate().centerOfMass()

distance = Vector.distance(com_first_water, com_second_water)
print("\nThe distance between the centers of mass of the waters is %s A" % distance)
```

Sire is divided into lots of libraries. You import the library you want to use using

from Sire.??? import *

or

import Sire.???

```
# Import the Sire Molecule library. This library contains
# everything needed to represent and manipulate atoms
# and molecules
from Sire.Mol import *

# Import the Sire Input/Output library. This library contains
# everything needed to load and save things to and from files
from Sire.IO import *

# Import the Sire Maths library. This library contains mathematical
# objects, e.g. vectors, angles, planes etc.
```

```
# Use the PDB object from Sire.IO to load a PDB file
# containing a water dimer
water_dimer = PDB().read("input/water_dimer.pdb")
```

```
# water_dimer is a container containing the two
# water molecules in the dimer. Lets get hold of
# each water molecule
first_water = water_dimer[MolIdx(0)].molecule()
print("The first water molecule is %s" % first_water)

# Now the second water molecule
second_water = water_dimer[MolIdx(1)].molecule()
print("The second water molecule is %s" % second_water)

# Let's print out all of the atoms
# of the first water
print("\nFirst water molecule:")
for i in range(0, first_water.nAtoms()):
    atom = first_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )


# Let's do the same to the second water
print "\nSecond water molecule:"
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's calculate the distance between the center of masses
# of the two water molecules
com_first_water = first_water.evaluate().centerOfMass()
com_second_water = second_water.evaluate().centerOfMass()

distance = Vector.distance(com_first_water, com_second_water)
print("\nThe distance between the centers of mass of the waters is %s A" % distance)
```

You can read molecules from PDB files using "PDB().read(filename)". This reads all molecules.

If you want to read just the first molecule, use molecule = PDB().readMolecule(filename)

```
# Import the Sire Molecule library. This library contains
# everything needed to represent and manipulate atoms
# and molecules
from Sire.Mol import *

# Import the Sire Input/Output library. This library contains
# everything needed to load and save things to and from files
from Sire.IO import *

# Import the Sire Maths library. This library contains mathematical
# objects, e.g. vectors, angles, planes etc.
from Sire.Maths import *

# Use the PDB object from Sire.IO to load a PDB file
# containing a water dimer
water_dimer = PDB().read("input/water_dimer.pdb")

# water_dimer is a container containing the two
# water molecules in the dimer. Lets get hold of
# each water molecule
first_water = water_dimer[MolIdx(0)].molecule()
print("The first water molecule is %s" % first_water)
# Now the second water molecule
second_water = water_dimer[MolIdx(1)].molecule()
print("The second water molecule is %s" % second_water)
# Let's print out all of the data about all of the atoms
# of the first water
print("\nFirst water molecule:")
for i in range(0, first_water.nAtoms()):
    atom = first_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's do the same to the second water
print "\nSecond water molecule:"
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's calculate the distance between the center of masses
# of the two water molecules
com_first_water = first_water.evaluate().centerOfMass()
com_second_water = second_water.evaluate().centerOfMass()

distance = Vector.distance(com_first_water, com_second_water)
print("\nThe distance between the centers of mass of the waters is %s A" % distance)
```

The molecules are held in a container, here called "water_dimer". The molecules can be identified by their Molecule Number (MolNum) or their Molecule Index (MolIdx) in the container

```
# Import the Sire Molecule library. This library contains
# everything needed to represent and manipulate atoms
# and molecules
from Sire.Mol import *

# Import the Sire Input/Output library. This library contains
# everything needed to load and save things to and from files
from Sire.IO import *

# Import the Sire Maths library. This library contains mathematical
# objects, e.g. vectors, angles, planes etc.
from Sire.Maths import *

# Use the PDB object from Sire.IO to load a PDB file
# containing a water dimer
water_dimer = PDB().read("input/water_dimer.pdb")

# water_dimer is a container containing the two
# water molecules in the dimer. Lets get hold of
# each water molecule
first_water = water_dimer[MolIdx(0)].molecule()
print("The first water molecule is %s" % first_water)

# Now the second water molecule
second_water = water_dimer[MolIdx(1)].molecule()
print("The second water molecule is %s" % second_water)

# Let's print out all of the data about all of the atoms
# of the first water
print("\nFirst water molecule:")
for i in range(0, first_water.nAtoms()):
    atom = first_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )


# Let's do the same to the second water
print "\nSecond water molecule:"
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )


# Let's calculate the distance between the centers of mass
# of the two water molecules
com_first_water = first_water.evaluate().centerOfMass()
com_second_water = second_water.evaluate().centerOfMass()

distance = Vector.distance(com_first_water, com_second_water)
print("\nThe distance between the centers of mass of the two water molecules is %s A" % distance)
```

The molecules are held in a container, here called "water_dimer". The molecules can be identified by their Molecule Number (MolNum) or their Molecule Index (MolIdx) in the container

MolNum is a unique number given to each loaded molecule (starting at 1 for first loaded molecule, and increasing by 1 for each loaded molecule).

MolIdx is the index in the container, e.g. the first molecule in the container is MolIdx(0), the second is MolIdx(1) etc.

A Molecule is a container for Atoms. We can get the atom using its name (AtomName), its number (AtomNum) or its index in the molecule (AtomIdx). Here we use molecule.nAtoms() to get the number of atoms, and then get hold of each one using its AtomIdx.

Using the name is also possible, e.g. oxygen = first_water.atom(AtomName("O00"))

```
# Let's print out all of the data about all of the atoms
# of the first water
print("\nFirst water molecule:")
for i in range(0, first_water.nAtoms()):
    atom = first_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )
```

The name of the atom is given by "atom.name()".

Other properties of the atom can be obtained by using "atom.property(....)", where you pass in the name of the property you want.

"element" gives the atom's element (C, O, N etc.)
"coordinates" gives the atom's coordinates

Use "atom.propertyKeys" to return a full list of all available properties.

```
# Let's do the same to the second water
print("\nSecond water molecule:")
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )
```

```
# Import the Sire Molecule library. This library contains
# everything needed to represent and manipulate atoms
# and molecules
from Sire.Mol import *

# Import the Sire Input/Output library. This library contains
# everything needed to load and save things to and from files
from Sire.IO import *
```

Next, we find the centers of mass of each water. You evaluate things like centers of mass using the ".evaluate()" function of a molecule or atom. This can be used to evaluate many values, e.g.

.centerOfMass() = center of mass

.centerOfGeometry() = center of geometry

.charge() = total charge

.mass() = total mass

.principalAxes() = principal axes

```
# Let's do the same to the second water
print "\nSecond water molecule:"
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
```

```
# Let's calculate the distance between the center of masses
# of the two water molecules
com_first_water = first_water.evaluate().centerOfMass()
com_second_water = second_water.evaluate().centerOfMass()
```

```
distance = Vector.distance(com_first_water, com_second_water)
print("\nThe distance between the centers of mass of the waters is %s A" % distance)
```

```
# Import the Sire Molecule library. This library contains
# everything needed to represent and manipulate atoms
# and molecules
from Sire.Mol import *

# Import the Sire Input/Output library. This library contains
# everything needed to load and save things to and from files
from Sire.IO import *

# Import the Sire Maths library. This library contains mathematical
# objects, e.g. vectors, angles, planes etc.
from Sire.Maths import *

# Use the PDB object from Sire.IO to load a PDB file
# containing a water dimer
water_dimer = PDB().read("input/water_dimer.pdb")

# water_dimer is a container containing the two
# water molecules in the dimer. Lets get hold of
# each water molecule
first_water = water_dimer[MolIdx(0)].molecule()
print("The first water molecule is %s" % first_water)

# Now the second water molecule
second_water = water_dimer[MolIdx(1)].molecule()
print("The second water molecule is %s" % second_water)

# Let's print out all of the data about all of the atoms
# of the first water
print("\nFirst water molecule:")
for i in range(0, first_water.nAtoms()):
    atom = first_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ))

# Let's do the same to the second water
print "\nSecond water molecule:"
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ))

# Let's calculate the distance between the center of masses
# of the two water molecules
com_first_water = first_water.evaluate().centerOfMass()
com_second_water = second_water.evaluate().centerOfMass()

distance = Vector.distance(com_first_water, com_second_water)
print("\nThe distance between the centers of mass of the waters is %s A" % distance)
```

The two centers of mass were returned as Vector objects. A Vector is a 3D point in space. You can calculate the distance between Vectors using Vector.distance(point_1, point_2)

Calculate angles using Vector.angle(point_1, point_2, point_3)

Calculate dihedrals using Vector.dihedral(point_1, point_2, point_3, point_4)

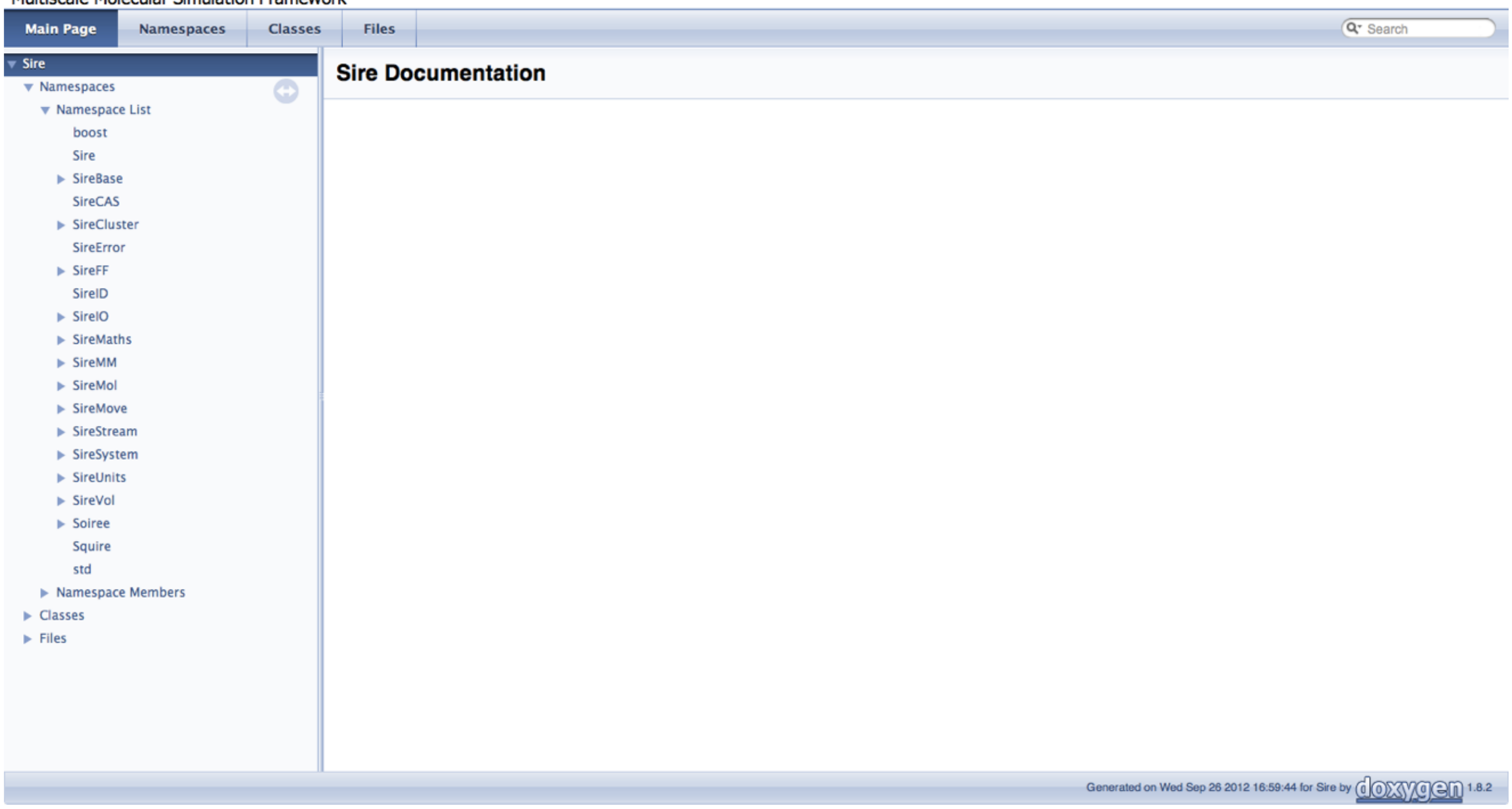
A simple Sire script: water_dimer.py

```
# Import the Sire Molecule library. This library contains
# everything needed to represent and manipulate atoms
# and molecules
from Sire.Mol import *

# Import the Sire Input/Output library. This library contains
# everything needed to load and save things to and from files
from Sire.IO import *

# Import the Sire Maths library. This library contains mathematical
# objects, e.g. vectors, angles, planes etc.
from Sire.Maths import *

# Use the PDB object from Sire.IO to load a PDB file
# containing a water dimer
water_dimer = PDB().read("input/water_dimer.pdb")

# water_dimer is a container containing the two
# water molecules in the dimer. Lets get hold of
# each water molecule
first_water = water_dimer[MolIdx(0)].molecule()
print("The first water molecule is %s" % first_water)

# Now the second water molecule
second_water = water_dimer[MolIdx(1)].molecule()
print("The second water molecule is %s" % second_water)

# Let's print out all of the data about all of the atoms
# of the first water
print("\nFirst water molecule:")
for i in range(0, first_water.nAtoms()):
    atom = first_water.atom(AtomIdx(i))
    print("%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's do the same to the second water
print "\nSecond water molecule:"
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom(AtomIdx(i))
    print(“%s: %s %s" % ( atom.name(),
                          atom.property("element"),
                          atom.property("coordinates") ) )

# Let's calculate the distance between the center of masses
# of the two water molecules
com_first_water = first_water.evaluate().centerOfMass()
com_second_water = second_water.evaluate().centerOfMass()

distance = Vector.distance(com_first_water, com_second_water)
print("\nThe distance between the centers of mass of the waters is %s A" % distance)
```

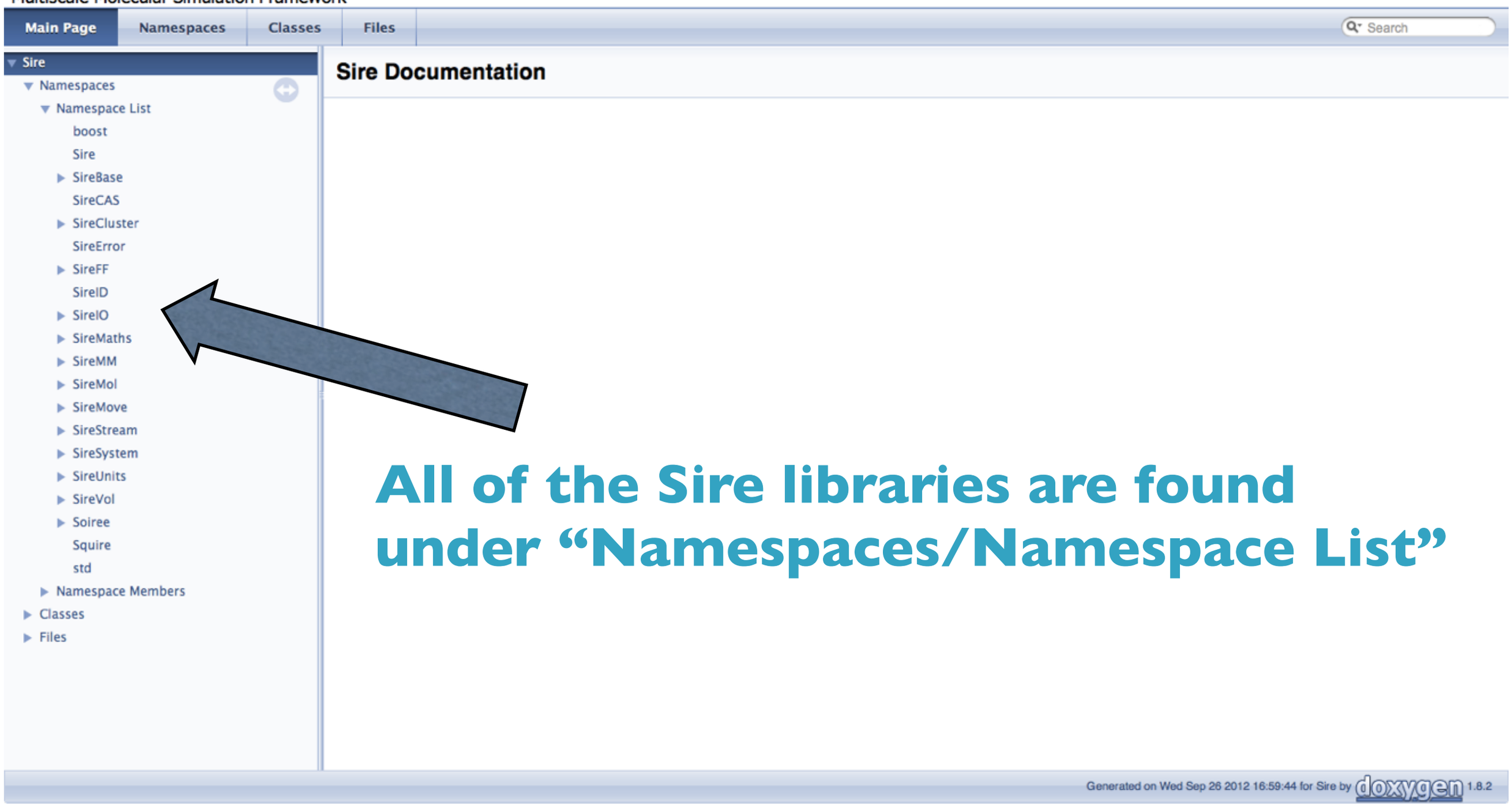
You can run the script using sire.app/bin/python

```
:-> ~/sire.app/bin/python water_dimer.py
"/Users/chris/sire.app/share/Sire/scripts/python.py"
Starting /Users/chris/sire.app/bin/python: number of threads equals 4
The first water molecule is Molecule(  : 1 : 10 : UID == {8aa41f91-fbcb-4129-b0ba-96e5547ff47f} )
The second water molecule is Molecule(  : 2 : 10 : UID == {67005db0-c042-445a-9750-141787474a49} )

First water molecule:
AtomName('O00'): Oxygen (O, 8) ( 14.986, -16.18, -11.971 )
AtomName('H01'): Hydrogen (H, 1) ( 14.813, -16.72, -11.2 )
AtomName('H02'): Hydrogen (H, 1) ( 14.693, -15.304, -11.721 )
AtomName('M03'): dummy (Xx, 0) ( 14.926, -16.137, -11.84 )

Second water molecule:
AtomName('O00'): Oxygen (O, 8) ( 18.337, -17.553, -16.079 )
AtomName('H01'): Hydrogen (H, 1) ( 18.529, -17.598, -17.016 )
AtomName('H02'): Hydrogen (H, 1) ( 17.805, -16.763, -15.982 )
AtomName('M03'): dummy (Xx, 0) ( 18.294, -17.457, -16.187 )

The distance between the centers of mass of the waters is 5.553492522321603 A
:->
```

Sire API documentation is at; http://siremol.org/apidocs/sire



Sire API documentation is at; http://siremol.org/apidocs/sire



All of the Sire libraries are found under “Namespaces/Namespace List”

Sire API documentation is at; http://siremol.org/apidocs/sire

Sire

Multiscale Molecular Simulation Framework

Main Page | Namespaces | Classes | Files | Search

Namespace List | Namespace Members

- Sire
  - Namespaces
    - Namespace List
      - boost
      - Sire
      - SireBase
      - SireCAS
      - SireCluster
      - SireError
      - SireFF
      - SireID
      - SireIO
      - SireMaths
      - SireMM
      - SireMol
      - SireMove
      - SireStream
      - SireSystem
      - SireUnits
      - SireVol
      - Soiree
      - Squire
      - std
    - Namespace Members
  - Classes
  - Files

Namespaces | Classes | Typedefs | Functions

SireMol Namespace Reference

Namespaces

namespace detail

Classes

class AngleID
class Atom
class AtomProperty< Vector >
class AtomMatcher
class AtomsIn
class Bead
class BeadEditor
class BeadID
class BeadIdx
class Beading
class MoleculeBeading

SireMol

Generated on Wed Sep 26 2012 16:59:34 for Sire by doxygen 1.8.2

All of the Sire libraries are found under “Namespaces/Namespace List”

Sire API documentation is at; http://siremol.org/apidocs/sire

Sire

Multiscale Molecular Simulation Framework

Main Page | Namespaces | Classes | Files

Molecule

Namespace List | Namespace Members

- Sire
  - Namespaces
    - Namespace List
      - boost
      - Sire
      - SireBase
      - SireCAS
      - SireCluster
      - SireError
      - SireFF
      - SireID
      - SireIO
      - SireMaths
      - SireMM
      - SireMol
      - SireMove
      - SireStream
      - SireSystem
      - SireUnits
      - SireVol
      - Soiree
      - Squire
      - std
    - Namespace Members
  - Classes
  - Files

Molecule
Molecule SireMol
molecule.cpp
molecule.h
molecule_metadata
moleculeAt SireMol::MoleculeGroup
MoleculeBeading
MoleculeBeading SireMol
MoleculeConstraint
MoleculeConstraint SireSystem
moleculeconstraint.cpp
moleculeconstraint.h
moleculecutting.cpp
moleculecutting.h

SireMol Namespace Reference

Namespaces

namespace detail

Classes

class AngleID
class Atom
class AtomProperty< Vector >
class AtomID
class AtomIdentifier
class AtomIdx
class AtomMatcher
class AtomIdxMatcher
class AtomNameMatcher
class BeadEditor
class BeadID
class BeadIdx
class Beading
class MoleculeBeading
class ResidueBeading

Or you can search for classes directly using the search box here, e.g. lets find the documentation for Sire.Mol.Molecule

SireMol

Generated on Wed Sep 26 2012 16:59:34 for Sire by doxygen 1.8.2

Sire API documentation is at; http://siremol.org/apidocs/sire

Sire

Multiscale Molecular Simulation Framework

Main Page | Namespaces | Classes | Files | Search

Class List | Class Index | Class Hierarchy | Class Members

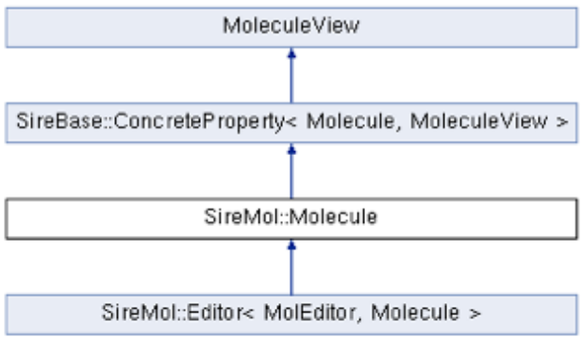
- GeometryPerturbations
- BondPerturbation
- AnglePerturbation
- DihedralPerturbation
- GroupAtomIDBase
- GroupAtomID
- GroupGroupID
- ImproperID
- MGID
- MGIdentifier
- MGIDsAndMaps
- MGIdx
- MGName
- MGNum
- MolAtomID
- Molecule
- MoleculeData
- MoleculeGroup
- MolGroupsBase
- MoleculeGroups
- MoleculeInfoData
- Molecules
- MoleculeView
- MolEditor
- MolStructureEditor
- MolID
- MolIdentifier
- MolIdx
- MolInfo
- MolName

SireMol::Molecule Class Reference

Public Member Functions | Static Public Member Functions | Protected Member Functions | Friends | List of all members

`#include <molecule.h>`

Inheritance diagram for SireMol::Molecule:



Public Member Functions

	Molecule ()
	Molecule (const QString &molname)
	Molecule (const **MoleculeData** &moldata)
	Molecule (const **Molecule** &other)
	~Molecule ()
Molecule &	**operator=** (const **Molecule** &other)
Molecule *	**clone** () const
QString	**toString** () const
bool	**isEmpty** () const
bool	**selectedAll** () const
AtomSelection	**selection** () const
const **MolName** &	**name** () const
MolNum	**number** () const
quint64	**version** () const

SireMol | Molecule

Generated on Wed Sep 26 2012 16:59:36 for Sire by doxygen 1.8.2

Adding TIP4P parameters to the water dimer water_parameters.py

```
from Sire.Mol import *
from Sire.MM import *
from Sire.IO import *
from Sire.Units import *
import Sire.Stream

water_dimer = PDB().read("input/water_dimer.pdb")

first_water = water_dimer[ MolIdx(0) ]
second_water = water_dimer[ MolIdx(1) ]

oxygen = first_water.atom( AtomName("O00") )
oxygen = oxygen.edit().setProperty("LJ", LJParameter( 3.15363*angstrom,  \
                                                      0.1550*kcal_per_mol) )

first_water = oxygen.molecule().commit()

print("Available properties of the first water molecule:")
print(first_water.propertyKeys())
print("The LJ property of the \"O00\" atom of the first water molecule:")
print(first_water.atom( AtomName("O00") ).property("LJ"))

second_water = second_water.atom( AtomName("O00") ) \
                           .edit().setProperty("LJ", LJParameter( 3.15363*angstrom, \
                                                                  0.1550*kcal_per_mol) ) \
                           .molecule().commit()

first_water = first_water.edit() \
                         .atom( AtomName("M03") ) \
                         .setProperty("charge", -1.04*mod_electron) \
                         .molecule().atom( AtomName("H01") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().atom( AtomName("H02") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().commit()

second_water = second_water.edit() \
                           .setProperty("charge", first_water.property("charge") ) \
                           .commit()

print("\nParameters of the atoms in the second water molecule:")
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom( AtomIdx(i) )
    print("%s : %s, %s" % (atom.name(), atom.property("charge"), atom.property("LJ")))

print("\nSaving the parameterised water molecules to disk...")
Sire.Stream.save( (first_water,second_water), "water_dimer.s3" )
```

```
from Sire.MM import *
from Sire.Units import *
import Sire.Stream
```

New Sire libraries in this script are;

Sire.Units

Contains definitions of all physical constants and dimensions (lengths, energies etc.)

Sire.Stream

Used to stream (load/save) Sire objects to and from datastreams. These can be passed over a network or saved or loaded from disk.

Sire.MM

Contains classes used to define molecular mechanics (MM) parameters and calculate MM energies and forces

```
from Sire.Mol import *
from Sire.MM import *
from Sire.IO import *
from Sire.Units import *

water_dimer = PDB().read("input/water_dimer.pdb")

first_water = water_dimer[ MolIdx(0) ]
second_water = water_dimer[ MolIdx(1) ]

oxygen = first_water.atom( AtomName("O00") )
oxygen = oxygen.edit().setProperty("LJ", LJParameter( 3.15363*angstrom,  \
                                                      0.1550*kcal_per_mol) )

first_water = oxygen.molecule().commit()

print("\nAvailable properties of the first water molecule:")
print(first_water.propertyKeys())
print("The LJ property of the \"O00\" atom of the first water molecule:")
print(first_water.atom( AtomName("O00") ).property("LJ"))

second_water = second_water.atom( AtomName("O00") ) \
                           .edit().setProperty("LJ", LJParameter( 3.15363*angstrom, \
                                                                  0.1550*kcal_per_mol) ) \
                           .molecule().commit()

first_water = first_water.edit() \
                         .atom( AtomName("M03") ) \
                         .setProperty("charge", -1.04*mod_electron) \
                         .molecule().atom( AtomName("H01") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().atom( AtomName("H02") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().commit()

second_water = second_water.edit() \
                           .setProperty("charge", first_water.property("charge") ) \
                           .commit()

print("\nParameters of the atoms in the second water molecule:")
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom( AtomIdx(i) )
    print("%s : %s, %s" % (atom.name(), atom.property("charge"), atom.property("LJ")))

print("\nSaving the parameterised water molecules to disk...")
Sire.Stream.save( (first_water,second_water), "water_dimer.s3" )
```

We use the same code as the last script to load the water dimer and get hold of the two water molecules...

```
from Sire.Mol import *
from Sire.MM import *
from Sire.IO import *
from Sire.Units import *
import Sire.Stream

water_dimer = PDB().read("input/water_dimer.pdb")

first_water = water_dimer[ MolIdx(0) ]
second_water = water_dimer[ MolIdx(1) ]

oxygen = first_water.atom( AtomName("O00") )
oxygen = oxygen.edit().setProperty("LJ", LJParameter( 3.15363*angstrom,  \
                                                      0.1550*kcal_per_mol) )

first_water = oxygen.molecule().commit()

print("Available properties of the first water molecule:")
print(first_water.propertyKeys())
print("The LJ property of the \"O00\" atom of the first water molecule:")
print(first_water.atom( AtomName("O00") ).property("LJ"))

second_water = second_water.atom( AtomName("O00") ) \
                           .edit().setProperty("LJ", LJParameter( 3.15363*angstrom, \
                                                                  0.1550*kcal_per_mol) ) \
                           .molecule().commit()

first_water = first_water.edit() \
                         .atom( AtomName("M03") ) \
                         .setProperty("charge", -1.04*mod_electron) \
                         .molecule().atom( AtomName("H01") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().atom( AtomName("H02") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().commit()

second_water = second_water.edit() \
                           .setProperty("charge", first_water.property("charge") ) \
                           .commit()

print("\nParameters of the atoms in the second water molecule:")
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom( AtomIdx(i) )
    print("%s : %s, %s" % (atom.name(), atom.property("charge"), atom.property("LJ")))

print("\nSaving the parameterised water molecules to disk...")
Sire.Stream.save( (first_water,second_water), "water_dimer.s3" )
```

Molecules are containers for atoms. We want to add Lennard Jones parameters to the oxygen atom of the first water. To do this, was first take a copy of that atom. We know that the name of the atom is "O00", so we can get it using "first_water.atom(AtomName("O00"))"

```
from Sire.Mol import *
from Sire.MM import *
from Sire.IO import *
from Sire.Units import *
import Sire.Stream

water_dimer = PDB().read("input/water_dimer.pdb")

first_water = water_dimer[ MolIdx(0) ]
second_water = water_dimer[ MolIdx(1) ]

oxygen = first_water.atom( AtomName("O00") )
oxygen = oxygen.edit().setProperty("LJ",
                    LJParameter( 3.15363*angstrom,  \
                                 0.1550*kcal_per_mol) )

first_water = oxygen.molecule().commit()

print("Available properties of the first water molecule:")
print(first_water.propertyKeys())
print("The LJ property of atom 0 of the first water molecule")
print(first_water.atom( AtomName("O00") ).property("LJ"))

second_water = second_water.edit().atom( AtomName("O00") )
                .edit().setProperty("LJ", LJParameter( 3.15363*angstrom, \
                                                       0.1550*kcal_per_mol) )
                .molecule().atom( AtomName("H01") ) \
                .setProperty("charge", 0.52*mod_electron) \
                .molecule().atom( AtomName("H02") ) \
                .setProperty("charge", 0.52*mod_electron) \
                .molecule().commit()

print("\nParameters of the atoms in the second water molecule:")
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom( AtomIdx(i) )
    print("%s : %s, %s" % (atom.name(), atom.property("charge"), atom.property("LJ")))

print("\nSaving the parameterised water molecules to disk...")
Sire.Stream.save( (first_water,second_water), "water_dimer.s3" )
```

To edit an atom or molecule, you need to call the ".edit()" function. From here, you can then use ".setProperty(...)" to set a property of that atom or molecule, e.g. here setting the "LJ" property to equal the passed LJ parameter.

Note that the LJ parameters in the LJParameter object must be dimensioned, i.e. you need to specify the units of length and energy.

When you have finished editing, you have to call the ".commit()" function to save your changes. You also need to update "first_water" to use the updated molecule (as editing "oxygen" edits a copy of the molecule)

```
first_water = oxygen.molecule().commit()

print("Available properties of the first water molecule:")
print(first_water.propertyKeys())
print("The LJ property of the \"O00\" atom of the first water molecul
```

Once edited, you can check that your edits were correct using the code here. Here we print the set of property keys (which should now contain "LJ"), and we print the LJ parameters for the atom "O00". Note that the other atoms in water will automatically be given dummy LJ parameters.

```
print("\nSaving the parameterised water molecules to disk...")
Sire.Stream.save( (first_water,second_water), "water_dimer.s3" )
```

```
from Sire.Mol import *
from Sire.MM import *
from Sire.IO import *
from Sire.Units import *
import Sire.Stream

water_dimer = PDB().read("input/water_dimer.pdb")
```

Next, we do the same thing to "second_water". In this case, selecting the atom, editing it, returning to the molecule and committing the changes are all placed on a single line of Python.

```
print("The LJ property of the \"O00\" atom of the first water molecule:")
print(first_water.atom( AtomName("O00") ).property("LJ"))
```

```
second_water = second_water.atom( AtomName("O00") ) \
                           .edit().setProperty("LJ",
                       LJParameter( 3.15363*angstrom, \
                                    0.1550*kcal_per_mol) ) \
                           .molecule().commit()
```

```
first_water = first_water.edit() \
                         .atom( AtomName("M03") ) \
                         .setProperty("charge", -1.04*mod_electron) \
                         .molecule().atom( AtomName("H01") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().atom( AtomName("H02") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().commit()

second_water = second_water.edit() \
                           .setProperty("charge", first_water.property("charge") ) \
                           .commit()

print("\nParameters of the atoms in the second water molecule:")
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom( AtomIdx(i) )
    print("%s : %s, %s" % (atom.name(), atom.property("charge"), atom.property("LJ")))

print("\nSaving the parameterised water molecules to disk...")
Sire.Stream.save( (first_water,second_water), "water_dimer.s3" )
```

Next we add TIP4P charges to the atoms of first_water. Again, all of the edits are combined together into a single Python line. Note that you can edit multiple atoms in a single line. Just remember to use ".molecule()" after editing each atom, so that you can the move onto the next atom.

```
first_water = first_water.edit() \
                         .atom( AtomName("M03") ) \
                         .setProperty("charge", -1.04*mod_electron) \
                         .molecule().atom( AtomName("H01") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().atom( AtomName("H02") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().commit()
```

Note that you must specify the units of charge!
Electron charges are "mod_electron"

```
from Sire.Mol import *
from Sire.MM import *
from Sire.IO import *
from Sire.Units import *
import Sire.Stream

water_dimer = PDB().read("input/water_dimer.pdb")

first_water = water_dimer[ MolIdx(0) ]
second_water = water_dimer[ MolIdx(1) ]

oxygen = first_water.atom( AtomName("O00") )
oxygen = oxygen.edit().setProperty("LJ", LJParameter( 3.15363*angstrom,  \
                                                      0.1550*kcal_per_mol) )

first_water = oxygen.molecule().commit()

print("Available properties of the first water molecule:")
print(first_water.propertyKeys())
```

Now we need to set the charges of second_water. Rather than set the charges of the atoms individually, we can call ".edit()" on the whole molecule, and copy the "charge" property from first_water.

```
second_water = second_water.edit() \
                           .setProperty("charge", \
                               first_water.property("charge") ) \
                           .commit()

print("\nSaving the parameterised water molecules to disk...")
Sire.Stream.save( (first_water,second_water), "water_dimer.s3" )
```

```
from Sire.Mol import *
from Sire.MM import *
from Sire.IO import *
from Sire.Units import *
import Sire.Stream

water_dimer = PDB().read("input/water_dimer.pdb")

first_water = water_dimer[ MolIdx(0) ]
second_water = water_dimer[ MolIdx(1) ]

oxygen = first_water.atom( AtomName("O00") )
oxygen = oxygen.edit().setProperty("LJ", LJParameter( 3.15363*angstrom,  \
                                                      0.1550*kcal_per_mol) )

first_water = oxygen.molecule().commit()

print("Available properties of the first water molecule:")
print(first_water.propertyKeys())
```

To check everything is correct, here we loop over all of the atoms in second_water and print out each atoms charge and LJ parameters. Note that atoms that have not had these parameters set will have null (dummy) values assigned automatically.

```
print("\nParameters of the atoms in the second water molecule:")
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom( AtomIdx(i) )
    print("%s : %s, %s" % (atom.name(), atom.property("charge"), \
                           atom.property("LJ")) )
```

Finally, we save (stream) first_water and second_water to a “Sire Saved Stream” (s3) file. This saves both Python/C++ objects, so that they can be loaded up and used in another file.

All Sire objects can be saved to s3 files. You can save as many objects as you like, using;

```
Sire.Stream.save( (obj1,obj2....), “filename.s3” )
```

You can reload the objects using;

```
(obj1,obj2,...) = Sire.Stream.load(“filename.s3”)
```

```
print("\nParameters of the atoms in the second water molecule:")
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom( AtomIdx(i) )
    print("%s : %s, %s" % (atom.name(), atom.property("charge"), atom.property("LJ")))

print "\nSaving the parameterised water molecules to disk..."
Sire.Stream.save( (first_water,second_water), "water_dimer.s3" )
```

Adding TIP4P parameters to the water dimer water_parameters.py

```
from Sire.Mol import *
from Sire.MM import *
from Sire.IO import *
from Sire.Units import *
import Sire.Stream

water_dimer = PDB().read("input/water_dimer.pdb")

first_water = water_dimer[ MolIdx(0) ]
second_water = water_dimer[ MolIdx(1) ]

oxygen = first_water.atom( AtomName("O00") )
oxygen = oxygen.edit().setProperty("LJ", LJParameter( 3.15363*angstrom,  \
                                                      0.1550*kcal_per_mol) )

first_water = oxygen.molecule().commit()

print("Available properties of the first water molecule:")
print(first_water.propertyKeys())
print("The LJ property of the \"O00\" atom of the first water molecule:")
print(first_water.atom( AtomName("O00") ).property("LJ"))

second_water = second_water.atom( AtomName("O00") ) \
                           .edit().setProperty("LJ", LJParameter( 3.15363*angstrom, \
                                                                  0.1550*kcal_per_mol) ) \
                           .molecule().commit()

first_water = first_water.edit() \
                         .atom( AtomName("M03") ) \
                         .setProperty("charge", -1.04*mod_electron) \
                         .molecule().atom( AtomName("H01") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().atom( AtomName("H02") ) \
                         .setProperty("charge", 0.52*mod_electron) \
                         .molecule().commit()

second_water = second_water.edit() \
                           .setProperty("charge", first_water.property("charge") ) \
                           .commit()

print("\nParameters of the atoms in the second water molecule:")
for i in range(0, second_water.nAtoms()):
    atom = second_water.atom( AtomIdx(i) )
    print("%s : %s, %s" % (atom.name(), atom.property("charge"), atom.property("LJ")))

print("\nSaving the parameterised water molecules to disk...")
Sire.Stream.save( (first_water,second_water), "water_dimer.s3" )
```

```
:->
:-> ~/sire.app/bin/python water_parameters.py
"/Users/chris/sire.app/share/Sire/scripts/python.py"
Starting /Users/chris/sire.app/bin/python: number of threads equals 4
Available properties of the first water molecule:
['PDB-residue-name', 'icode', 'PDB-chain-name', 'formal-charge', 'PDB-atom-name', 'element', 'b-factor', 'coordinates',
 'LJ', 'PDB-segment-name']
The LJ property of the "O00" atom of the first water molecule:
LJ( sigma = 3.15363 A, epsilon = 0.155 kcal mol-1 )

Parameters of the atoms in the second water molecule:
AtomName('O00') : 0 |e|, LJ( sigma = 3.15363 A, epsilon = 0.155 kcal mol-1 )
AtomName('H01') : 0.52 |e|, LJ( sigma = 0 A, epsilon = 0 kcal mol-1 )
AtomName('H02') : 0.52 |e|, LJ( sigma = 0 A, epsilon = 0 kcal mol-1 )
AtomName('M03') : -1.04 |e|, LJ( sigma = 0 A, epsilon = 0 kcal mol-1 )

Saving the parameterised water molecules to disk...
:->
```

Calculating intermolecular energies
water_energy.py

```
from Sire.Maths import *
from Sire.MM import *

import Sire.Stream

(first_water, second_water) = Sire.Stream.load("water_dimer.s3")

cljff = InterCLJFF("clj")

cljff.add( first_water )
cljff.add( second_water )

total_nrg = cljff.energy()

print("The total interaction energy between the dimer is %s" % total_nrg)

coul_nrg = cljff.energy( cljff.components().coulomb() )
lj_nrg = cljff.energy( cljff.components().lj() )

print("The coulomb energy is %s. The LJ energy is %s." % (coul_nrg, lj_nrg))

com_vector = second_water.evaluate().centerOfMass() - \
             first_water.evaluate().centerOfMass()

com_distance = com_vector.length()
print("\nThe water molecules are separated by %s A" % com_distance)

com_vector = com_vector.normalise()
delta_vector = (2 - com_distance) * com_vector

print("\nTo move the water molecules to be separated by 2 A, we " \
      "need to translate the second water by %s" % delta_vector)

second_water = second_water.move().translate(delta_vector).commit()

com_distance = Vector.distance( first_water.evaluate().centerOfMass(),
                                second_water.evaluate().centerOfMass() )

print("The updated distance is indeed %s A" % com_distance)

cljff.update(second_water)

print(cljff.energies())

print "\nDistance   Energies"
for i in range(0,10):
    second_water = second_water.move().translate( 0.2*com_vector ).commit()
    cljff.update(second_water)
    print("%f  %s" % ( Vector.distance( first_water.evaluate().centerOfMass(), \
                                        second_water.evaluate().centerOfMass() ),
                       cljff.energies() ) )
```

```
from Sire.Maths import *
from Sire.MM import *
import Sire.Stream

(first_water, second_water) =
Sire.Stream.load("water_dimer.s3")
```

We first import the required Sire libraries.

We then load the parameterised Sire Molecule objects representing “first_water” and “second_water” from the Sire s3 file.

```
com_vector = com_vector.normalise()
delta_vector = (2 - com_distance) * com_vector

print("\nTo move the water molecules to be separated by 2 A, we " \
      "need to translate the second water by %s" % delta_vector)

second_water = second_water.move().translate(delta_vector).commit()

com_distance = Vector.distance( first_water.evaluate().centerOfMass(),
                                second_water.evaluate().centerOfMass() )

print("The updated distance is indeed %s A" % com_distance)

cljff.update(second_water)

print(cljff.energies())

print "\nDistance   Energies"
for i in range(0,10):
    second_water = second_water.move().translate( 0.2*com_vector ).commit()
    cljff.update(second_water)
    print("%f  %s" % ( Vector.distance( first_water.evaluate().centerOfMass(), \
                                        second_water.evaluate().centerOfMass() ),
                       cljff.energies() ) )
```

```
from Sire.Maths import *
from Sire.MM import *

import Sire.Stream

cljff = InterCLJFF("clj")

cljff.add( first_water )
cljff.add( second_water )

total_nrg = cljff.energy()
```

Sire uses "ForceField" objects to calculate energies and forces. A ForceField is a special container for molecules. It calculates energies and forces for all of the contained molecules.

Here we create a ForceField to calculate the intermolecular coulomb and LJ energy (InterCLJFF). We add first_water and second_water to this ForceField container.

Calling .energy() calculates the total energy for all molecules contained in the ForceField.

The energy of a ForceField can be made up of several sub-components. For example, the CLJ ForceField objects have both .lj() and .coulomb() components.

```
coul_nrg = cljff.energy( cljff.components().coulomb() )
lj_nrg = cljff.energy( cljff.components().lj() )

print("The coulomb energy is %s. The LJ energy is %s."  \
                                        (coul_nrg, lj_nrg) )
```

You can get the value of these components using;

ff.energy(ff.components().component())

e.g.

cljff.energy(cljff.components().coulomb())

Next, we want to calculate the energy as a function of the distance between the centers of mass of the two water molecules (from 2-4 A). To start, we must get the vector between the two centers of mass, and then work out by how much to translate the second water so that it is 2 A away from the first...

```
com_vector = second_water.evaluate().centerOfMass() - \
             first_water.evaluate().centerOfMass()
com_distance = com_vector.length()
print("\nThe water molecules are separated by %s A" % com_distance)

com_vector = com_vector.normalise()
delta_vector = (2 - com_distance) * com_vector

print("\nTo move the water molecules to be separated by 2 A, we " \
      "need to translate the second water by %s" % delta_vector)
```

To actually move the molecule, we need to call the ".move()" function. This is like the ".edit()" function. It allows you to perform a series of moves on a molecule (or atom). At the end of the move(s) you must save the changes using ".commit()".

Here we use ".move().translate(...).commit()" to translate second_water. If we wanted to rotate the molecule, we would use ".move().rotate(...).commit()"

```
second_water = second_water.move().translate(delta_vector).commit()

com_distance =
Vector.distance( first_water.evaluate().centerOfMass(),
                 second_water.evaluate().centerOfMass() )

print("The updated distance is indeed %s A" % com_distance)
```

```
from Sire.Maths import *
from Sire.MM import *

import Sire.Stream
```

Remember! Moving a molecule is the same as editing a molecule. The move applies *only* to the copy of the molecule being ".move()"d. Moving second_water does not change the copy of second_water held in the "cljff" ForceField.

To update the copy of second_water held in cljff, you must call the ".update(...)" function.

Once you have called ".update(...)" you will see that the energies of the ForceField have changed.

```
print("The updated distance is indeed %s A" % com_distance)
```

```
cljff.update(second_water)
```

```
print cljff.energies()
```

```
    second_water = second_water.move().translate( 0.2*com_vector ).commit()
    cljff.update(second_water)
    print("%f  %s" % ( Vector.distance( first_water.evaluate().centerOfMass(), \
                                        second_water.evaluate().centerOfMass() ),
                       cljff.energies() ) )
```

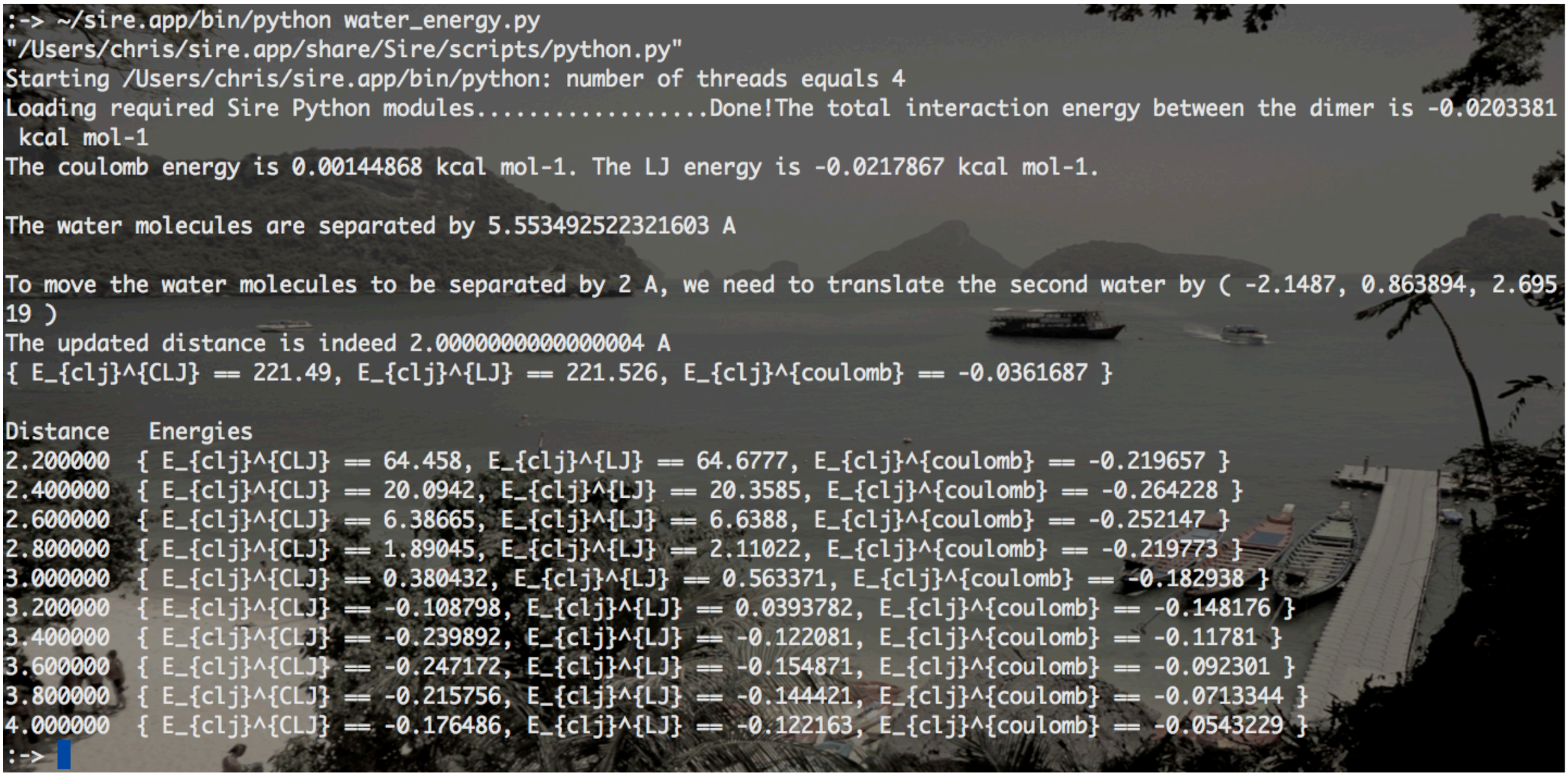
Finally, we loop over all 10 steps of 0.2 A between 2 A and 4 A, and we translate the second water molecule by 0.2 A along the vector between the water molecules center of mass.

After each move, we update the cljff ForceField, print the distance between the center of masses, and print all of the energy components of cljff (available via “cljff.energies()”)

```
print("\nDistance   Energies")
for i in range(0,10):
    second_water = second_water.move() \
                        .translate( 0.2*com_vector ).commit()
    cljff.update(second_water)
    print("%f  %s" % \
   ( Vector.distance( first_water.evaluate().centerOfMass(), \
                      second_water.evaluate().centerOfMass() ), \
     cljff.energies() ) )
```

Calculating intermolecular energies
water_energy.py

```
from Sire.Maths import *
from Sire.MM import *

import Sire.Stream

(first_water, second_water) = Sire.Stream.load("water_dimer.s3")

cljff = InterCLJFF("clj")

cljff.add( first_water )
cljff.add( second_water )

total_nrg = cljff.energy()

print("The total interaction energy between the dimer is %s" % total_nrg)

coul_nrg = cljff.energy( cljff.components().coulomb() )
lj_nrg = cljff.energy( cljff.components().lj() )

print("The coulomb energy is %s. The LJ energy is %s." % (coul_nrg, lj_nrg))

com_vector = second_water.evaluate().centerOfMass() - \
             first_water.evaluate().centerOfMass()

com_distance = com_vector.length()
print("\nThe water molecules are separated by %s A" % com_distance)

com_vector = com_vector.normalise()
delta_vector = (2 - com_distance) * com_vector

print("\nTo move the water molecules to be separated by 2 A, we " \
      "need to translate the second water by %s" % delta_vector)

second_water = second_water.move().translate(delta_vector).commit()

com_distance = Vector.distance( first_water.evaluate().centerOfMass(),
                                second_water.evaluate().centerOfMass() )

print("The updated distance is indeed %s A" % com_distance)

cljff.update(second_water)

print(cljff.energies())

print "\nDistance   Energies"
for i in range(0,10):
    second_water = second_water.move().translate( 0.2*com_vector ).commit()
    cljff.update(second_water)
    print("%f  %s" % ( Vector.distance( first_water.evaluate().centerOfMass(), \
                                        second_water.evaluate().centerOfMass() ),
                       cljff.energies() ) )
```


:-> ~/sire.app/bin/python water_energy.py
"/Users/chris/sire.app/share/Sire/scripts/python.py"
Starting /Users/chris/sire.app/bin/python: number of threads equals 4
Loading required Sire Python modules.....Done!The total interaction energy between the dimer is -0.0203381
kcal mol-1
The coulomb energy is 0.00144868 kcal mol-1. The LJ energy is -0.0217867 kcal mol-1.
The water molecules are separated by 5.553492522321603 A
To move the water molecules to be separated by 2 A, we need to translate the second water by (-2.1487, 0.863894, 2.695
19)
The updated distance is indeed 2.0000000000000004 A
{ E_{clj}^{CLJ} == 221.49, E_{clj}^{LJ} == 221.526, E_{clj}^{coulomb} == -0.0361687 }
Distance Energies
2.200000 { E_{clj}^{CLJ} == 64.458, E_{clj}^{LJ} == 64.6777, E_{clj}^{coulomb} == -0.219657 }
2.400000 { E_{clj}^{CLJ} == 20.0942, E_{clj}^{LJ} == 20.3585, E_{clj}^{coulomb} == -0.264228 }
2.600000 { E_{clj}^{CLJ} == 6.38665, E_{clj}^{LJ} == 6.6388, E_{clj}^{coulomb} == -0.252147 }
2.800000 { E_{clj}^{CLJ} == 1.89045, E_{clj}^{LJ} == 2.11022, E_{clj}^{coulomb} == -0.219773 }
3.000000 { E_{clj}^{CLJ} == 0.380432, E_{clj}^{LJ} == 0.563371, E_{clj}^{coulomb} == -0.182938 }
3.200000 { E_{clj}^{CLJ} == -0.108798, E_{clj}^{LJ} == 0.0393782, E_{clj}^{coulomb} == -0.148176 }
3.400000 { E_{clj}^{CLJ} == -0.239892, E_{clj}^{LJ} == -0.122081, E_{clj}^{coulomb} == -0.11781 }
3.600000 { E_{clj}^{CLJ} == -0.247172, E_{clj}^{LJ} == -0.154871, E_{clj}^{coulomb} == -0.092301 }
3.800000 { E_{clj}^{CLJ} == -0.215756, E_{clj}^{LJ} == -0.144421, E_{clj}^{coulomb} == -0.0713344 }
4.000000 { E_{clj}^{CLJ} == -0.176486, E_{clj}^{LJ} == -0.122163, E_{clj}^{coulomb} == -0.0543229 }
:->

Performing a Monte Carlo simulation on the water dimer in a periodic boundaries simulation box

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *

import Sire.Stream

(first_water, second_water) = Sire.Stream.load("water_dimer.s3")

cljff = InterCLJFF("cljff")
cljff.add(first_water)
cljff.add(second_water)

system = System()
system.add(cljff)

lam = Symbol("lambda")
total_nrg = cljff.components().lj() + lam * cljff.components().coulomb()

system.setComponent( system.totalComponent(), total_nrg )
system.setConstant( lam, 0.5 )

mobile_mols = MoleculeGroup("mobile_molecules")
mobile_mols.add(first_water)
mobile_mols.add(second_water)

system.add(mobile_mols)

space = PeriodicBox( Vector(5,5,5) )
system.setProperty( "space", space )
system.add( SpaceWrapper(Vector(0), mobile_mols) )

move = RigidBodyMC(mobile_mols)
move.setTemperature( 25*celsius )
move.setMaximumTranslation(0.5 * angstrom)
move.setMaximumRotation(5 * degrees)

moves = WeightedMoves()
moves.add( move, 1 )

print("Running 100 moves...")
new_system = moves.move(system, 100, True)

# Now lets run a simulation, writing out a PDB trajectory
PDB().write(system.molecules(), "output000.pdb")
print(system.energies())

for i in range(1,11):
    system = moves.move(system, 1000, True)
    print("%d: %s" % (i, system.energies()))
    PDB().write(system.molecules(), "output%003d.pdb" % i)

print("Move information:")
print(moves)
```

```
from Sire.MM import *
from Sire.Mol import *
from Sire.Vol import *
from Sire.Maths import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *
import Sire.Stream

(first_water, second_water) = Sire.Stream.load("water_dimer.s3")

cljff = InterCLJFF("cljff")
cljff.add(first_water)
cljff.add(second_water)

system = System()
system.add(cljff)

lam = Symbol("lambda")
total_nrg = cljff.components().lj() + lam * cljff.components().coulomb()

system.setComponent( system.totalComponent(), total_nrg )
system.setConstant( lam, 0.5 )

mobile_mols = MoleculeGroup("mobile molecules")
mobile_mols.add(first_water)
mobile_mols.add(second_water)

system.add(mobile_mols)

space = PeriodicBox( Vector(5,5,5) )
system.setProperty( "space", space )
system.add( SpaceWrapper(Vector(0), mobile_mols) )

move = RigidBodyMC(mobile_mols)
move.setTemperature(25 * celsius)
move.setMaximumTranslation(0.5 * angstrom)
move.setMaximumRotation(5 * degrees)

moves = WeightedMoves()
moves.add( move, 1 )

print("Running 100 moves...")
new_system = moves.move(system, 100, True)

# Now lets run a simulation, writing out a PDB trajectory
PDB().write(system.molecules(), "output000.pdb")
print(system.energies())

for i in range(1,11):
    system = moves.move(system, 100, True)
    print("%d: %s" % (i, system.energies()))
    PDB().write(system.molecules(), "output%003d.pdb" % i)

print("Move information:")
print(moves)
```

Sire.Vol

Everything needed to define spaces (volumes), e.g. periodic boundary boxes

Sire.Move

Everything needed to perform Monte Carlo and Molecular Dynamics moves

Sire.System

Everything needed to package together molecules and forcefields into systems

Sire.CAS

A complete Computer Algebra System

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *

import Sire.Stream

(first_water, second_water) =
Sire.Stream.load("water_dimer.s3")

cljff = InterCLJFF("cljff")
cljff.add(first_water)
cljff.add(second_water)
```

We first load the water dimer from the .s3 file, create the cljff intermolecular coulomb and Lennard Jones forcefield and add both water molecules to that forcefield.

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *

import Sire.Stream

(first_water, second_water) = Sire.Stream.load("water_dimer.s3")

cljff = InterCLJFF("cljff")
cljff.add(first_water)
cljff.add(second_water)

system = System()
system.add(cljff)
```

Next, we create a System(). System objects allow all of the molecules, forcefields etc. needed to represent a complete simulation system to be packaged together into a single object. In this case, our System will contain the cljff ForceField, and the Molecules contained in that ForceField (our water dimer).

A Sire simulation takes the form of constructing a System object, and then applying moves on that System.

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *

import Sire.Stream

(first_water, second_water) = Sire.Stream.load("water_dimer.s3")

cljff = InterCLJFF("cljff")
cljff.add(first_water)
cljff.add(second_water)

lam = Symbol("lambda")
total_nrg = cljff.components().lj() + \
            lam * cljff.components().coulomb()
```

Now, we use the Computer Algebra System to create an algebraic expression that is used to compute the total energy of the System. In this case, we will create a Symbol called “lambda”, and say that the total energy is the LJ energy from cljff, plus lambda times the coulomb energy. This allows us to use lambda to turn on and off the coulomb energy, by scaling it between 0 and 1.

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *
```

We then tell the system to use the "total_nrg" expression we have just defined to calculate the ".totalComponent()" total energy of the system. We then set lambda equal to 0.5.

```
system.setComponent( system.totalComponent(), total_nrg )
system.setConstant( lam, 0.5 )
```

Note that we can add as many components as we want to "system", e.g.

```
system.setComponent( Symbol("E_coul"), \
    lam * cljff.components().coulomb() )
```

Different components of different ForceFields can be combined together in any way you want. This gives you a lot of freedom to create interesting Hamiltonians for free energy sims.

```
print("Move information:")
print(moves)
```

Next, we need to create a group of molecules in the system. A "MoleculeGroup" is a container for molecules that allows them to be grouped together. In this case, we want to use the group to control which molecules will be moved.

```
lam = Symbol("lambda")
total_nrg = cljff.components().lj() + lam * cljff.components().coulomb()
mobile_mols = MoleculeGroup("mobile_molecules")
mobile_mols.add(first_water)
mobile_mols.add(second_water)

system.add(mobile_mols)
```

system.add(mobile_mols)

```
system.setProperty( "space", space )
system.add( SpaceWrapper(Vector(0), mobile_mols) )

move = RigidBodyMC(mobile_mols)
move.setTemperature( 25*celsius )
move.setMaximumTranslation(0.5 * angstrom)
move.setMaximumRotation(5 * degrees)

moves = WeightedMoves()
moves.add( move, 1 )

print("Running 100 moves...")
new_system = moves.move(system, 100, True)

# Now lets run a simulation, writing out a PDB trajectory
PDB().write(system.molecules(), "output000.pdb")
print(system.energies())

for i in range(1,11):
    system = moves.move(system, 1000, True)
    print("%d: %s" % (i, system.energies()))
    PDB().write(system.molecules(), "output%003d.pdb" % i)

print("Move information:")
print(moves)
```

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.System import *
from Sire.CAS import *
import Sire.Stream
```

We create a PeriodicBox object that provides a periodic boundaries simulation space. We initialise the box to have dimension 5Ax5Ax5A.

System objects have properties just like molecules and atoms. Here, we set the “space” property equal to the periodic box.

```
mobile_mols = MoleculeGroup("mobile_molecules")
mobile_mols.add(first_water)
mobile_mols.add(second_water)

space = PeriodicBox( Vector(5,5,5) )
system.setProperty( "space", space )
system.add( SpaceWrapper(Vector(0), mobile_mols) )

move = RigidBodyMC(mobile_mols)
move.setTemperature( 25*celsius )
move.setMaximumTranslation(0.5 * angstrom)
```

To ensure that moves on molecules are wrapped back into the periodic box, here we add a “SpaceWrapper” constraint. This wraps molecules in the “mobile_mols” MoleculeGroup back into the periodic box centered at the origin.

```
moves = WeightedMoves()
moves.add(move)

print("Running 100 moves...")

# Now lets run a simulation, writing out a PDB trajectory
print(system.energies())

for i in range(1,101):
    moves.move(system, 1000, True)
    print("%d: %s" % (i, system.energies()))
    PDB().write(system.molecules(), "output%003d.pdb" % i)

print("Move information:")
print(moves)
```

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *

import Sire.Stream

(first_water, second_water) = Sire.Stream.load("water_dimer.s3")
cljff = InterCLJFF("cljff")
cljff.add(first_water)
cljff.add(second_water)
system = System()
system.add(cljff)
lam = Symbol("lambda")
total_nrg = cljff.components().lj() + lam * cljff.components().coulomb()
system.setComponent( system.totalComponent(), total_nrg )
system.setConstant( lam, 0.5 )
mobile_mols = MoleculeGroup("mobile_molecules")
mobile_mols.add(first_water)
mobile_mols.add(second_water)
system.add(mobile_mols)
space = PeriodicBox(Vector(5,5,5))
system.setProperty( "space", space )
system.add( SpaceWrapper(Vector(0), mobile_mols) )

move = RigidBodyMC(mobile_mols)
move.setTemperature( 25*celsius )
move.setMaximumTranslation(0.5 * angstrom)
move.setMaximumRotation(5 * degrees)
moves = WeightedMoves()
moves.add( move, 1 )
new_system = moves.move(system, 100, True)

# Now lets run a simulation, writing out a PDB trajectory
PDB().write(system.molecules(), "output000.pdb")
print(system.energies())

for i in range(1,11):
    system = moves.move(system, 1000, True)
    print("%d: %s" % (i, system.energies()))
    PDB().write(system.molecules(), "output%003d.pdb" % i)

print("Move information:")
print(moves)
```

With the system now complete, we can construct the Move objects that will perform moves on that System. Here, we create a RigidBodyMC move that performs rigid body translation and rotation Monte Carlo moves on the molecules in "mobile_mols". We set the temperature and maximum move deltas.

Just as a System groups together all of the forcefields, molecules etc. to be simulated, so a "Moves" object groups together all of the individual Move objects to be performed.

Here we use a WeightedMoves object. This collects together each move to be applied to the system, randomly choosing which move to apply next based on its weight. Here, we have just a single move (the RigidBodyMC move), so we leave the weight as 1. If we have more moves (e.g. InternalMove), we could add that with a different weight.

```
moves = WeightedMoves()
moves.add( move, 1 )

# Now lets run a simulation, writing out a PDB trajectory
PDB().write(system.molecules(), "output000.pdb")
print(system.energies())

for i in range(1,11):
    system = moves.move(system, 1000, True)
    print("%d: %s" % (i, system.energies()))
    PDB().write(system.molecules(), "output%003d.pdb" % i)

print("Move information:")
print(moves)
```

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *

import Sire.Stream

(first_water, second_water) = Sire.Stream.load("water_dimer.s3")

cljff = InterCLJFF("cljff")
```

You perform the moves by calling the ".move(...)" function. You pass in the system you want to move, the number of moves you want to perform, and whether or not you want to collect thermodynamic statistics (e.g. free energies).

Note that the moves are performed on a *copy* of the system. The updated system after the moves have been performed is returned by the function.

```
print("Running 100 moves...")
new_system = moves.move(system, 100, True)
PDB().write(system.molecules(), "output000.pdb")
print(system.energies())

for i in range(1,11):
    system = moves.move(system, 1000, True)
    print("%d: %s" % (i, system.energies()))
    PDB().write(system.molecules(), "output%003d.pdb" % i)

print("Move information:")
print(moves)
```

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *

import Sire.Stream
```

Now, we run a simulation, printing the energies and saving a PDB of the system every 1000 moves. We perform 10 blocks of 1000 moves.

Note that PDB().write(molecules, filename) is being used to write the PDB files. Note also that "system.energies()" returns the values of all energy components of the system.

```
# Now lets run a simulation, writing out a PDB trajectory
PDB().write(system.molecules(), "output000.pdb")
print(system.energies())

for i in range(1,11):
    system = moves.move(system, 1000, True)
    print("%d: %s" % (i, system.energies()))
    PDB().write(system.molecules(), "output%003d.pdb" % i)
```

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *

import Sire.Stream

(first_water, second_water) = Sire.Stream.load("water_dimer.s3")

cljff = InterCLJFF("cljff")
cljff.add(first_water)
cljff.add(second_water)

system = System()
system.add(cljff)

lam = Symbol("lambda")
total_nrg = cljff.components().lj() + lam * cljff.components().coulomb()

system.setComponent( system.totalComponent(), total_nrg )
system.setConstant( lam, 0.5 )

mobile_mols = MoleculeGroup("mobile_molecules")
mobile_mols.add(first_water)
mobile_mols.add(second_water)

system.add(mobile_mols)

space = PeriodicBox( Vector(5,5,5) )
system.setProperty( "space", space )
```

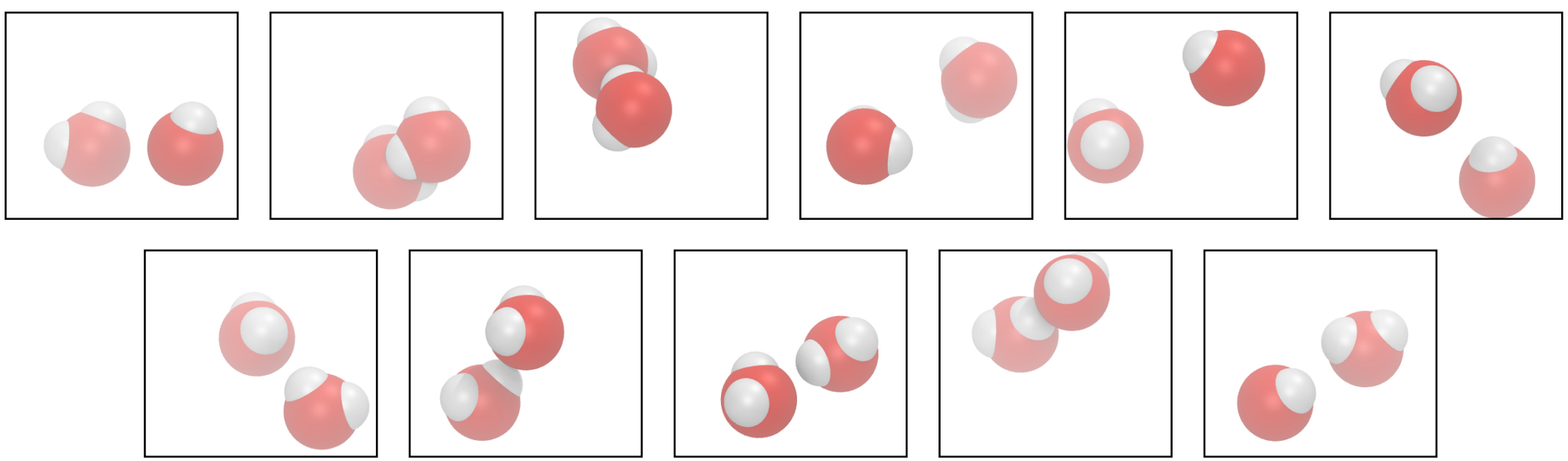
Finally(!) we print out all of the statistics about the moves that have been performed, e.g. the number of attempted, accepted and rejected MC moves. This is useful when you want to optimise the move deltas to get reasonable acceptance ratios.

```
print("Move information:")
print(moves)
```

Performing a Monte Carlo simulation on the water dimer in a periodic boundaries simulation box

```
from Sire.MM import *
from Sire.Mol import *
from Sire.IO import *
from Sire.Maths import *
from Sire.Units import *
from Sire.Vol import *
from Sire.Move import *
from Sire.System import *
from Sire.CAS import *

import Sire.Stream

(first_water, second_water) = Sire.Stream.load("water_dimer.s3")

cljff = InterCLJFF("cljff")
cljff.add(first_water)
cljff.add(second_water)

system = System()
system.add(cljff)

lam = Symbol("lambda")
total_nrg = cljff.components().lj() + lam * cljff.components().coulomb()

system.setComponent( system.totalComponent(), total_nrg )
system.setConstant( lam, 0.5 )

mobile_mols = MoleculeGroup("mobile_molecules")
mobile_mols.add(first_water)
mobile_mols.add(second_water)

system.add(mobile_mols)

space = PeriodicBox( Vector(5,5,5) )
system.setProperty( "space", space )
system.add( SpaceWrapper(Vector(0), mobile_mols) )

move = RigidBodyMC(mobile_mols)
move.setTemperature( 25*celsius )
move.setMaximumTranslation(0.5 * angstrom)
move.setMaximumRotation(5 * degrees)

moves = WeightedMoves()
moves.add( move, 1 )

print("Running 100 moves...")
new_system = moves.move(system, 100, True)

# Now lets run a simulation, writing out a PDB trajectory
PDB().write(system.molecules(), "output000.pdb")
print(system.energies())

for i in range(1,11):
    system = moves.move(system, 1000, True)
    print("%d: %s" % (i, system.energies()))
    PDB().write(system.molecules(), "output%003d.pdb" % i)

print("Move information:")
print(moves)
```

```
:-> ~/sire.app/bin/python water_moves.py
"/Users/chris/sire.app/share/Sire/scripts/python.py"
Starting /Users/chris/sire.app/bin/python: number of threads equals 4
Loading required Sire Python modules.....Done!Running 100 moves...
{ E_{cljff}^{CLJ} == 10.85, E_{cljff}^{LJ} == 20.3237, E_{cljff}^{coulomb} == -9.47373, E_{total} == 15.5868 }
1: { E_{cljff}^{CLJ} == -0.304133, E_{cljff}^{LJ} == -0.106163, E_{cljff}^{coulomb} == -0.19797, E_{total} == -0.205148
 }
2: { E_{cljff}^{CLJ} == -2.81326, E_{cljff}^{LJ} == -0.129886, E_{cljff}^{coulomb} == -2.68337, E_{total} == -1.47157 }
3: { E_{cljff}^{CLJ} == -4.00464, E_{cljff}^{LJ} == 0.0629801, E_{cljff}^{coulomb} == -4.06762, E_{total} == -1.97083 }
4: { E_{cljff}^{CLJ} == -2.90169, E_{cljff}^{LJ} == 0.138657, E_{cljff}^{coulomb} == -3.04035, E_{total} == -1.38152 }
5: { E_{cljff}^{CLJ} == -4.49171, E_{cljff}^{LJ} == 0.802804, E_{cljff}^{coulomb} == -5.29452, E_{total} == -1.84445 }
6: { E_{cljff}^{CLJ} == -3.19366, E_{cljff}^{LJ} == 0.112444, E_{cljff}^{coulomb} == -3.3061, E_{total} == -1.54061 }
7: { E_{cljff}^{CLJ} == -0.849621, E_{cljff}^{LJ} == -0.0726853, E_{cljff}^{coulomb} == -0.776936, E_{total} == -0.4611
53 }
8: { E_{cljff}^{CLJ} == -3.24727, E_{cljff}^{LJ} == 0.372409, E_{cljff}^{coulomb} == -3.61968, E_{total} == -1.43743 }
9: { E_{cljff}^{CLJ} == -5.33381, E_{cljff}^{LJ} == 0.324787, E_{cljff}^{coulomb} == -5.65859, E_{total} == -2.50451 }
10: { E_{cljff}^{CLJ} == -4.68523, E_{cljff}^{LJ} == 0.122259, E_{cljff}^{coulomb} == -4.80749, E_{total} == -2.28148 }
Move information:
WeightedMoves{
  1 : weight == 1
      RigidBodyMC( maximumTranslation() = 0.5 A, maximumRotation() = 5 degrees nAccepted() = 4889 nRejected() = 5211 )
}
:->
```



Sire Saved Streams (.s3) files make excellent restart files. You can save a simulation using;

```
import Sire.Stream
Sire.Stream.save( (system,moves), “restart_file.s3” )
```

You can then reload the restart file and continue the simulation using;

```
import Sire.Stream
(system, moves) = Sire.Stream.load(“restart_file.s3”)

system = moves.move(system, 10000, True)
```

Sire will automatically load any Sire libraries needed by the .s3 files, so you don’t need to do anything else yourself :-)

The restart files are portable and versioned too!

Thanks :-)

- If you want any more information, check out the website (http://siremol.org), the API documentation (http://siremol.org/apidocs/sire)
- Feel free to join the Sire User's mailing list or download Sire, from http://sire.googlecode.com