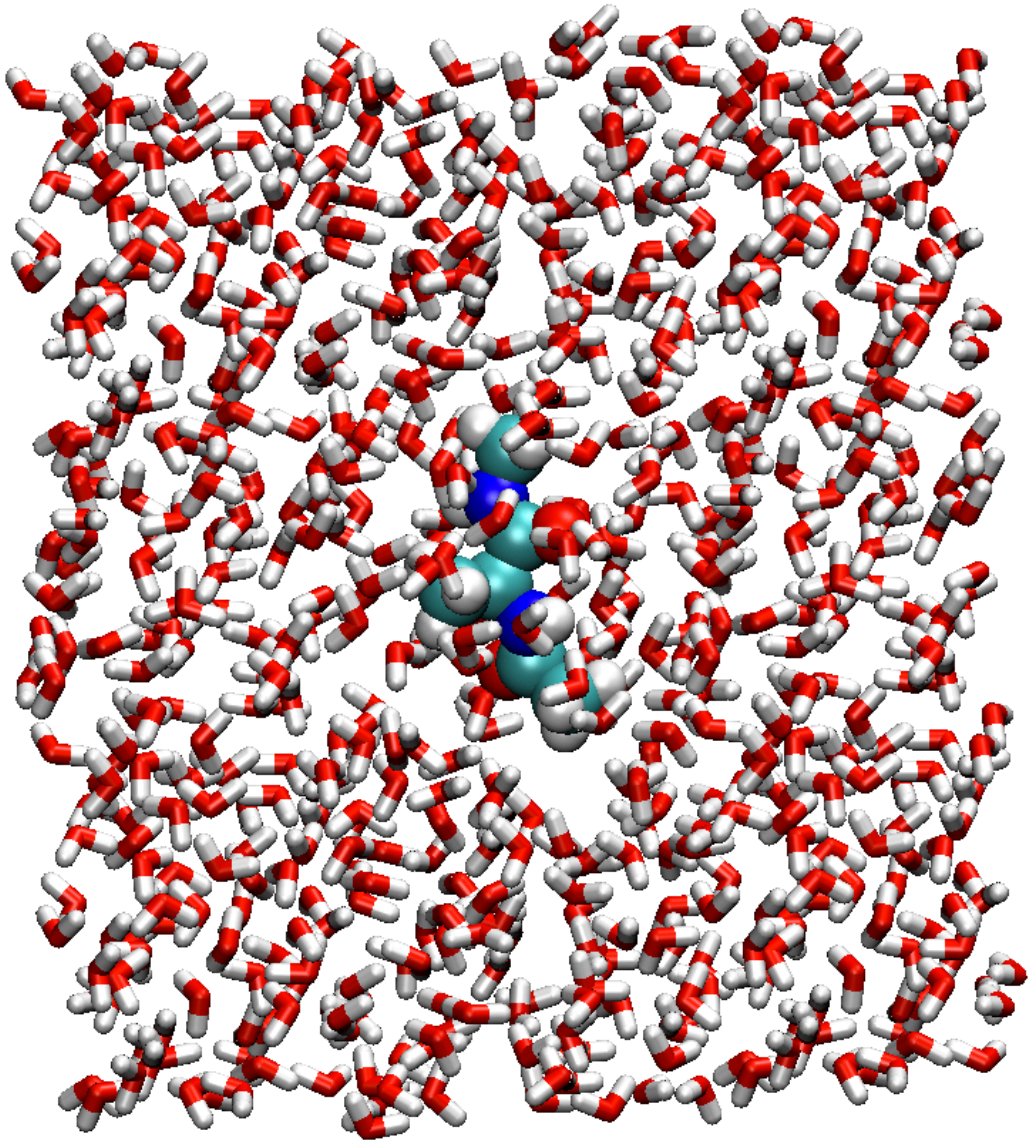
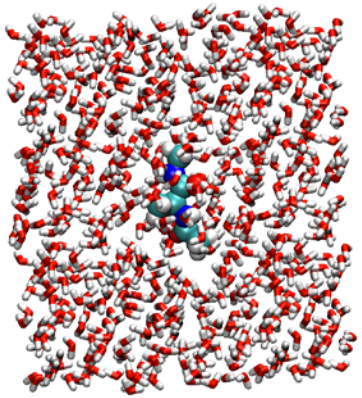
Exercise 1: Running a simulation using OpenMM python app
Alanine dipeptide in Water



Exercise 1: Running a simulation using OpenMM python app

Purpose: To check the OpenMM installation and to make you familiar with a typical OpenMM python script



```
from simtk.openmm.app import *
from simtk.openmm import *
from simtk.unit import *
from sys import stdout

# Reading structure and force field files
pdb = PDBFile('input_exercise1.pdb')
forcefield = ForceField('amber99sb.xml', 'tip3p.xml')

# Creating System
system = forcefield.createSystem(pdb.topology, nonbondedMethod=PME, nonbondedCutoff=1.0*nanometer,
      constraints=HBonds)
integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)

# Creating simulation context
simulation = Simulation(pdb.topology, system, integrator)
simulation.context.setPositions(pdb.positions)

# Minimizing System
simulation.minimizeEnergy(maxIterations=25)

# Adding Reporters
simulation.reporters.append(PDBReporter('output_exercise1.pdb', 5))
simulation.reporters.append(StateDataReporter(stdout, 100, step=True,
potentialEnergy=True, temperature=True))

# Running simulation
simulation.step(1000)
```

Exercise 1: Running a simulation using OpenMM python app

Run exercise using “python exercise1.py”

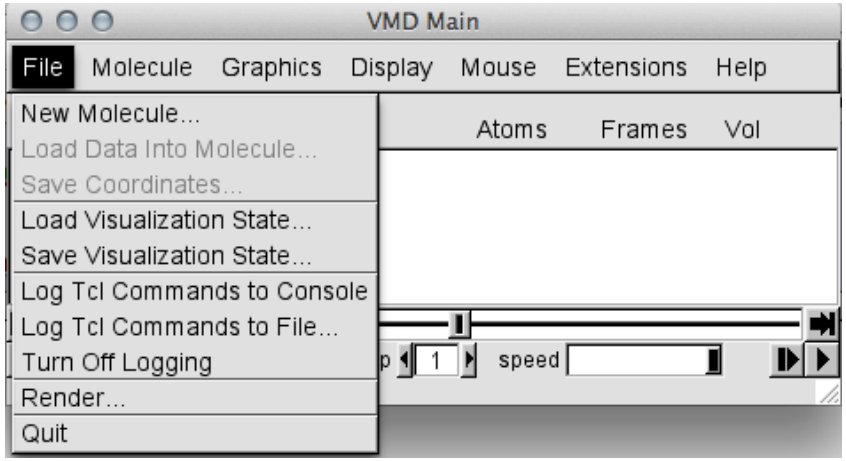
To fix the “Import error” or “Library not found”:

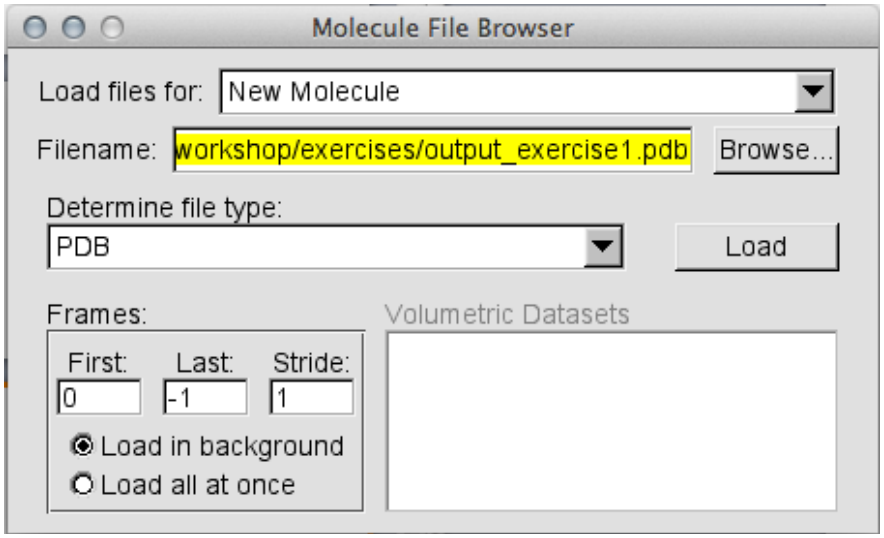
Mac OS X:
export DYLD_LIBRARY_PATH=/usr/local/openmm/lib:/usr/local/cuda/lib
Linux:
export LD_LIBRARY_PATH=/usr/local/openmm/lib:/usr/local/cuda/lib
Windows:
Refer to page 14 and 15 of the OpenMM Application Guide.

You should see the following output:

```
Creating System
Using Platform: OpenCL
Minimizing Energy
Adding Reporters to report Potential Energy and Temperature every 100 steps
Running Simulation for 1000 steps
Running step: 0
#"Step","Potential Energy (kJ/mole)","Temperature (K)"
100,-26643.6088239,175.727733927
Running step: 100
200,-26218.1442337,194.691558193
.
.
1000,-24957.7316677,283.859340959
Finished Simulation.
```

Exercise 1: Running a simulation using OpenMM python app
Viewing the output in VMD



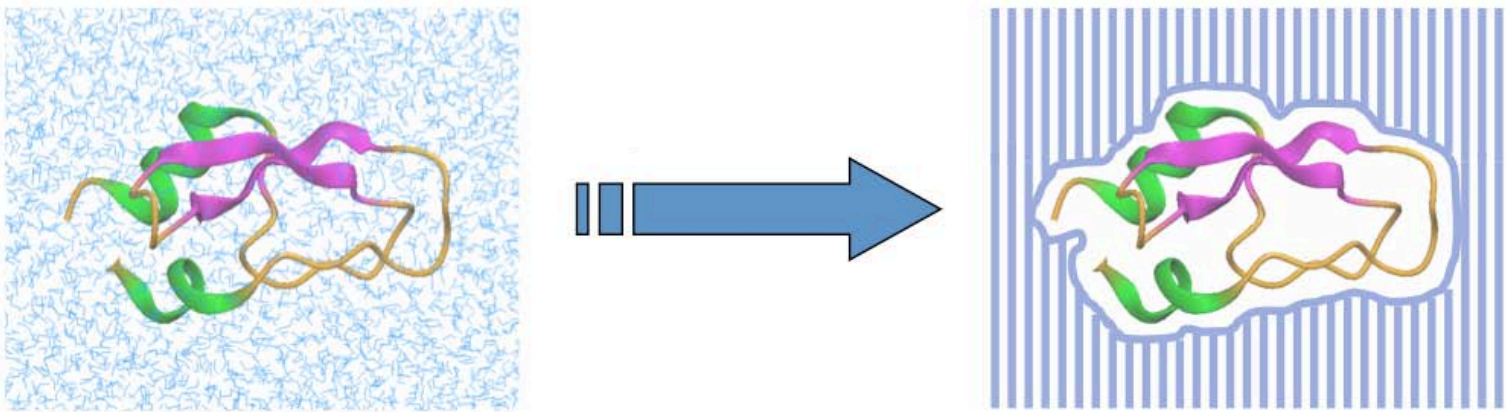


To load a new molecule, select New Molecule... from the File menu in the Main window, this will open the Files window.

We will load a PDB (Protein Data Bank) file output_exercise1.pdb containing the simulation results of the exercise1

Solvent models in OpenMM

```
forcefield = ForceField('amber99sb.xml', 'tip3p.xml')
```



$$H = H_{\text{prot}} + H_{\text{prot-wat}} + H_{\text{wat}}$$

$$H = H_{\text{prot}} + \Delta G_{\text{solv}}$$

Explicit water model

Implicit water model

The solvation free energy in principle contains all information of solvent on equilibrium properties of solute.

Important assumptions are required in order to efficiently estimate the solvation free energy without having to extensively sample all water degrees of freedom.

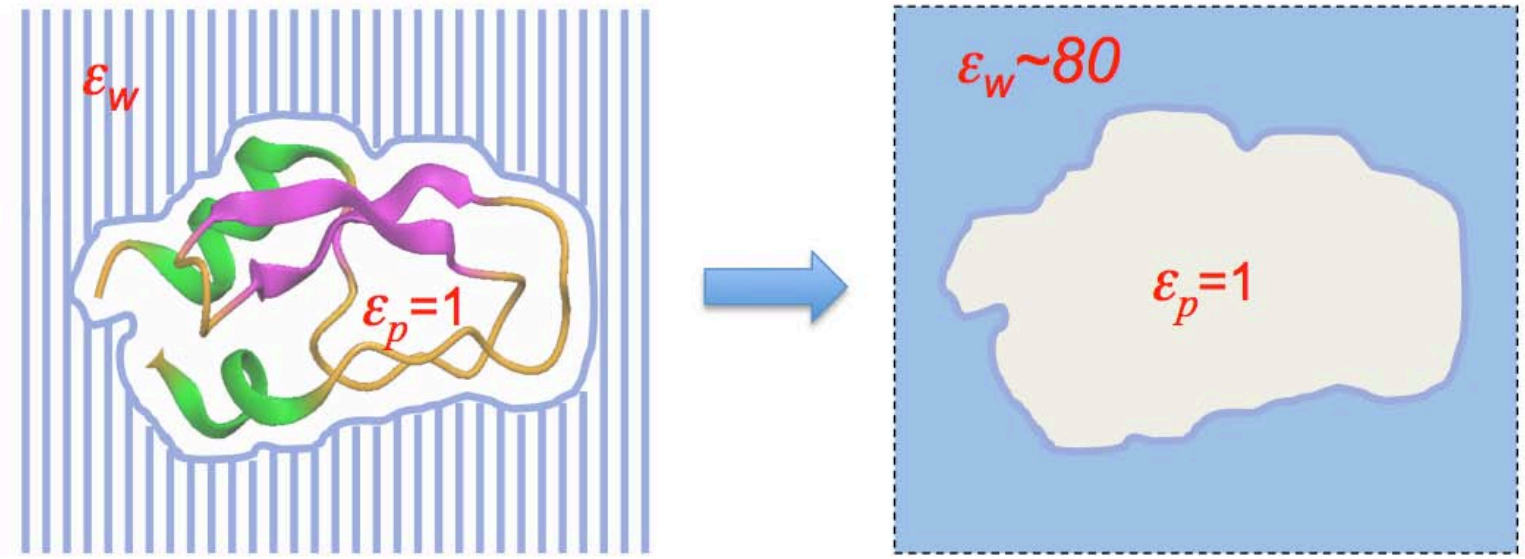
Implicit solvent models

Water dynamics is typically much faster than that of protein conformational diffusion. Thus, water can be described as a continuum medium.

$$\Delta G_{\text{solv}} = \Delta G_{\text{elec}} + \Delta G_{\text{np}} \qquad \Delta G_{\text{np}} = \gamma \cdot S$$

Generalized Born approximation for calculating the electrostatic component:

$$\Delta G_{\text{elec}} \approx \frac{1}{2}\left(\frac{1}{\varepsilon_{\text{w}}} - \frac{1}{\varepsilon_{\text{p}}}\right)\sum_{ij}\frac{q_i q_j}{\sqrt{r_{ij}^2 + R_i^{\text{GB}} R_j^{\text{GB}} \exp\left(-r_{ij}^2 / 4 R_i^{\text{GB}} R_j^{\text{GB}}\right)}}$$



Solvent models in OpenMM

File	Water Model
`tip3p.xml`	TIP3P water model[8]
`tip4pew.xml`	TIP4P-Ew water model[9]
`tip5p.xml`	TIP5P water model[10]
`spce.xml`	SPC/E water model[11]

File	Implicit Solvation Model
`amber96_obc.xml`	GBSA-OBC solvation model[12] for use with AMBER96 force field
`amber99_obc.xml`	GBSA-OBC solvation model for use with AMBER99 force fields
`amber03_obc.xml`	GBSA-OBC solvation model for use with AMBER03 force field
`amber10_obc.xml`	GBSA-OBC solvation model for use with AMBER10 force field
`amoeba2009_gk.xml`	Generalized Kirkwood solvation model[13] for use with AMOEBA force field

Integrators in OpenMM

integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)

Other Integrators:
For constant energy simulations: VerletIntegrator(0.002*picoseconds)

Brownian Dynamics: BrownianIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)

Variable time step: VariableLangevinIntegrator(300*kelvin, 1/picosecond, 0.001)
VariableVerletIntegrator(0.001)

Temperature coupling:
system.addForce(AndersenThermostat(300*kelvin, 1/picosecond))

Pressure Coupling:
system.addForce(MonteCarloBarostat(1*bar, 300*kelvin))

Constraints and Reporters in OpenMM

system = prmtop.createSystem(nonbondedMethod=NoCutoff, constraints=HBonds)

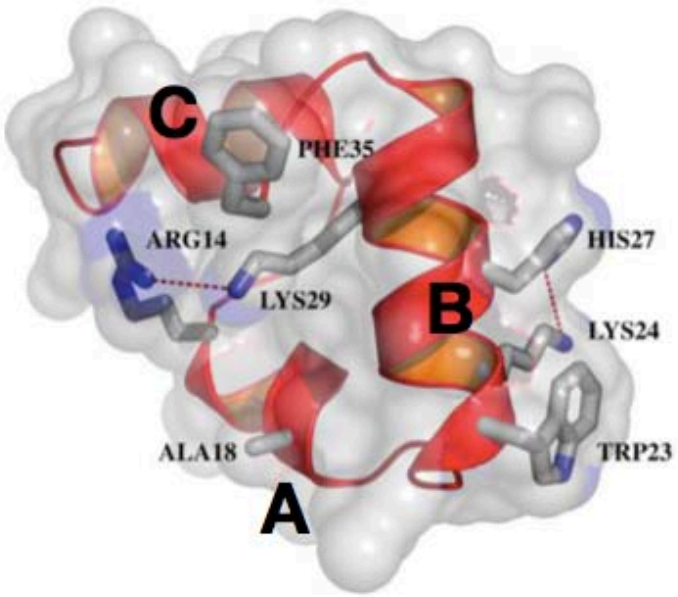
Value	Meaning
`None`	No constraints are applied. This is the default value.
`HBonds`	The lengths of all bonds that involve a hydrogen atom are constrained.
`AllBonds`	The lengths of all bonds are constrained.
`HAngles`	The lengths of all bonds are constrained. In addition, all angles of the form H-X-H or H-O-X (where X is an arbitrary atom) are constrained.

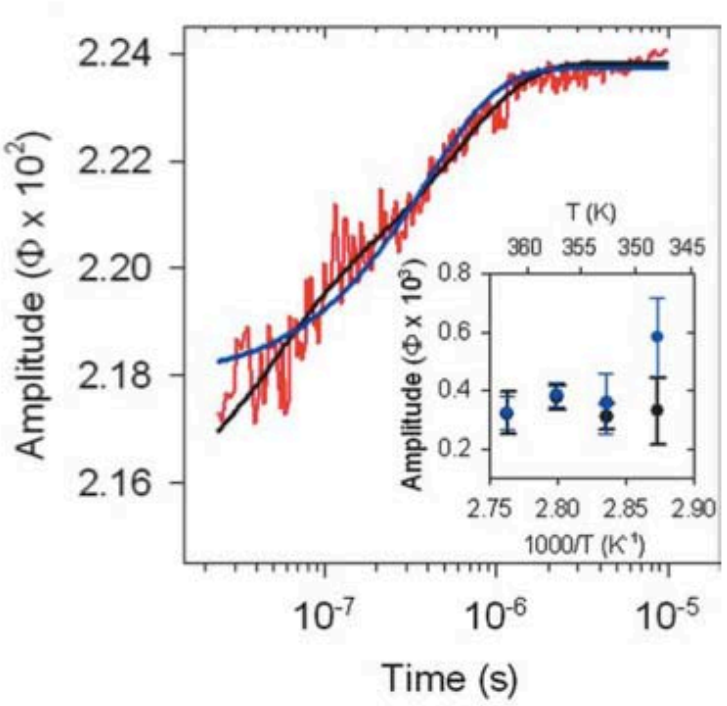
Reporters for printing values of State parameters during simulation:

simulation.reporters.append(StateDataReporter('data.txt', 1000, time=True, temperature=True, kineticEnergy=True, potentialEnergy=True, totalEnergy=True, volume=True, density=True, separator=' '))

Exercise 2: Simulation of villin headpiece in implicit solvent using OpenMM python app

Purpose: How to set up protein from a given pdb file and use implicit solvents in OpenMM





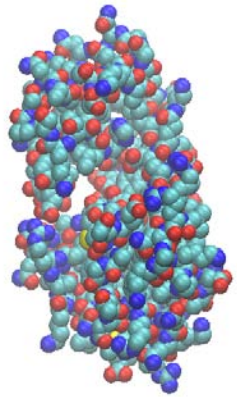
Copy exercise1.py to exercise2.py
Use input_exercise2.pdb as the starting structure.
Use amber99_obc as the implicit solvent model.
Change friction parameter to 91/ps for implicit solvent model.

Exercise 2: Simulating Villin in Implicit Solvent using OpenMM python app

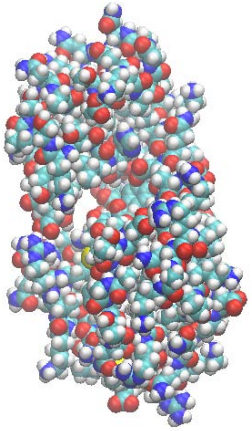
```
Error Message: “Template Not found”.

This message implies that there are missing atoms in the residues
of this Protein. We need to use modeller class to build missing
      atoms.
```

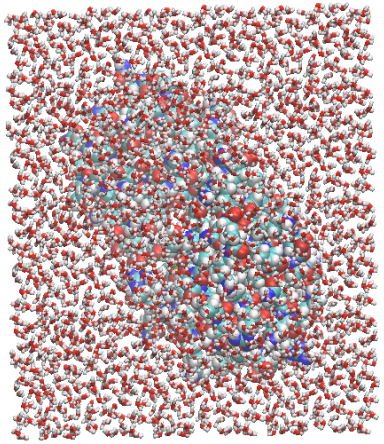
Model Building and Editing using OpenMM



Missing hydrogen atoms



Missing water box



Solvated system

OpenMM modeller class can fix these issues.

```
pdb = PDBFile('input.pdb')
modeller = Modeller(pdb.topology, pdb.positions)
# ... Call some modelling functions here ...
system = forcefield.createSystem(modeller.topology, nonbondedMethod=PME)
```

Available modeller functions:

Adding Hydrogen: modeller.addHydrogen(forcefield, pH=5.0)

Adding Solvent: modeller.addSolvent(forcefield, padding=1.0*nanometers, model='tip5p')
modeller.addSolvent(forcefield, boxSize=Vec3(5.0, 3.5, 3.5)*nanometers)

Adding Ions: modeller.addSolvent(forcefield, ionicStrength=0.1*molar, positiveIon='K+')

(refer OpenMM Application Guide section 4, page34)

Exercise 2: Simulating Villin in Implicit Solvent using OpenMM python app

Purpose: How to set up protein from a given pdb file and use implicit solvents in OpenMM

Copy exercise1.py to exercise2.py
Use input_exercise2.pdb as the starting structure.
Use amber99_obc as the implicit solvent model.
Change friction parameter to 91/ps for implicit solvent model.
Use Modeller class to add missing hydrogen atoms to protein.

```
pdb = PDBFile('input_exercise2.pdb')
forcefield = ForceField('amber99sb.xml', 'amber99_obc.xml')
print "Building Model.."
modeller = Modeller(pdb.topology, pdb.positions)
print('Adding hydrogens...')
modeller.addHydrogens(forcefield)
print "Creating System"
system = forcefield.createSystem(modeller.topology, constraints=HBonds)
integrator = LangevinIntegrator(300*kelvin, 91/picosecond, 0.002*picoseconds)
simulation = Simulation(modeller.topology, system, integrator)
print "Using Platform:", simulation.context.getPlatform().getName()
simulation.context.setPositions(modeller.positions)
```

Exercise 3: Simulation of villin headpiece in explicit solvent using OpenMM python app

Purpose: Use Modeller class to build a water box and use explicit solvent in OpenMM

Copy exercise1.py to exercise3.py
Use input_exercise2.pdb as the starting structure.
Use amber99sb and tip3p as the explicit solvent model.
Add explicit water molecules using the modeller class.

```
pdb = PDBFile('input_exercise3.pdb')
forcefield = ForceField('amber99sb.xml', 'tip3p.xml')
print "Building Model..."
modeller = Modeller(pdb.topology, pdb.positions)
print('Adding hydrogens...')
modeller.addHydrogens(forcefield)
print "Adding Water.."
modeller.addSolvent(forcefield, model='tip3p', padding=1.0*nanometers)
print "Creating System..."
system = forcefield.createSystem(modeller.topology, nonbondedMethod=PME,
nonbondedCutoff=1.0*nanometer, constraints=HBonds)
integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)
```