

Why write a script?

- Molecular dynamics simulations require a lot of information about
 - Input data
 - coordinates
 - topology
 - which atoms are included in energy terms
 - parameters for functions in energy terms
 - System description
 - periodicity
 - constraints
 - Integrators
 - algorithms to propagate forward in time
 - adjust box size
 - adjust kinetic energy (temperature)
- Simulation
 - how long to run
 - how much output data to store
- OpenMM can be run from widely-used computer programming languages, python and C++, facilitating extension and combination with other code

- This is the default script in the OpenMM script builder: <http://builder.openmm.org>
- Let's go through what the parameters mean and I'll give you values for a simple simulation of ubiquitin, a protein that we looked at during the VMD tutorial
- If you hover your mouse over the parameter names, it'll also give a brief description
- A more detailed description of simulation parameters is available at <http://docs.openmm.org/latest/userguide/application.html#simulation-parameters>

The screenshot shows the OpenMM Script Builder web interface. The top navigation bar includes 'OpenMM Script Builder', 'Get Help', and buttons for 'openmm.py' and 'Save Script'. The left sidebar has tabs for 'General', 'System', 'Integrator', and 'Simulation', with 'General' currently selected. The main panel displays various simulation parameters with their values:

- Input coordinates: input.pdb
- Input topology: input.prmtp
- Forcefield: AMBER99sb-ildn
- Water Model: TIP3P
- Platform: CUDA
- Precision: mixed
- Device index: (empty)
- OpenCL platform indx: (empty)

On the right, the generated Python script is displayed, starting with a header comment and followed by the simulation setup code:

```
#####
# this script was generated by openmm-builder. to customize it further,
# you can save the file to disk and edit it with your favorite editor.
#####

from __future__ import print_function
from simtk.openmm import app
import simtk.openmm as mm
from simtk import unit
from sys import stdout

pdb = app.PDBFile('input.pdb')
forcefield = app.ForceField('amber99sbildn.xml', 'tip3p.xml')

system = forcefield.createSystem(pdb.topology, nonbondedMethod=app.PME,
    nonbondedCutoff=1.0*unit.nanometers, constraints=app.HBonds, rigidWater=True,
    ewaldErrorTolerance=0.0005)
integrator = mm.LangevinIntegrator(300*unit.kelvin, 1.0/unit.picoseconds,
    2.0*unit.femtoseconds)
integrator.setConstraintTolerance(0.00001)

platform = mm.Platform.getPlatformByName('CUDA')
properties = {'CudaPrecision': 'mixed'}
simulation = app.Simulation(pdb.topology, system, integrator, platform,
    properties)
simulation.context.setPositions(pdb.positions)

print('Minimizing...')
simulation.minimizeEnergy()

simulation.context.setVelocitiesToTemperature(300*unit.kelvin)
print('Equilibrating...')
simulation.step(100)

simulation.reporters.append(app.DCDReporter('trajectory.dcd', 1000))
simulation.reporters.append(app.StateDataReporter(stdout, 1000, step=True,
    potentialEnergy=True, temperature=True, progress=True, remainingTime=True,
    speed=True, totalSteps=1000, separator='\t'))
```