Python + NEURON

Python + NEURON

```
>>> hoc.execute(’hoc statement’)
oc> nrnpython("python statement")
```

Python + NEURON

```
>>> hoc.execute('hoc statement')
oc> nrnpython("python statement")
```

- Native Hoc/Python interaction

Python + NEURON

```
>>> hoc.execute('hoc statement')
oc> nrnpython("python statement")
```

- Native Hoc/Python interaction
- Installation

Python + NEURON

```
>>> hoc.execute('hoc statement')
oc> nrnpython("python statement")
```

- Native Hoc/Python interaction
- Installation
- Numpy

Installation

Installation

```
>>> import neuron
```

Installation

```
>>> import neuron
```

Linux	i686 x86_64	
Mac	OS X	10.4 10.5
MSWin		

Installation

```
>>> import neuron
```

Linux	i686 x86_64		Python 2.3 2.4 2.5
Mac	OS X	10.4 10.5	
MSWin			

Installation

```
>>> import neuron
```

Linux	i686 x86_64		Python 2.3 2.4 2.5
Mac	OS X	10.4 10.5	
MSWin			

Launch NEURON
Python

Installation

```
>>> import neuron
```

Linux	i686 x86_64		Python 2.3 2.4 2.5
Mac	OS X	10.4 10.5	
MSWin	Cygwin MinGW		
Launch	NEURON Python		

Installation

```
>>> import neuron
```

Linux	i686 x86_64		Python 2.3 2.4 2.5
Mac	OS X	10.4 10.5	
MSWin	Cygwin MinGW		
Launch	NEURON Python		NumPy

sec	
	a = numpy.arange(0,10.0,0.00001)
0.02	b = numpy.array(a)
21.2	v = neuron.Vector(a)
43.7	a = numpy.array(v)

sec	
	a = numpy.arange(0,10.0,0.00001)
0.02	b = numpy.array(a)
21.2	v = neuron.Vector(a)
43.7	a = numpy.array(v)
0.01	b = numpy.array(a)
0.14	v = neuron.Vector(a)
0.07	b = v.to_python(numpy.zeros(v.size()))

sec	
	a = numpy.arange(0,10.0,0.00001)
0.02	b = numpy.array(a)
21.2	v = neuron.Vector(a)
43.7	a = numpy.array(v)
0.01	b = numpy.array(a)
0.14	v = neuron.Vector(a)
0.07	b = v.to_python(numpy.zeros(v.size()))
	a = range(0, 1000000)
0.16	b = numpy.array(a)
0.06	v = neuron.Vector(a)
0.04	b = v.to_python()

Python using Hoc

nrniv −python

Python using Hoc

```
nrniv −python
import neuron
h = neuron.h
print h
```

TopLevelHocInterpreter

Python using Hoc

```
nrniv −python
import neuron
h = neuron.h
print h
  TopLevelHocInterpreter
h('''
  a=5
  objref vec
  vec = new Vector()
  strdef s
  s = "string"
  func f() { return $1 * $1 }
''')
```

Python using Hoc

print h.a, h.vec, h.s, h.f(3)

5.0 Vector[0] string 9.0

Python using Hoc

print h.a, h.vec, h.s, h.f(3)

5.0 Vector[0] string 9.0

s = h.s

h.s = ’hello’

print s, h.s

string hello

Python using Hoc

print h.a, h.vec, h.s, h.f(3)
5.0 Vector[0] string 9.0

s = h.s

h.s = ’hello’

print s, h.s
string hello

f = h.f

print f, f(5)
f() 25.0

Python using Hoc

```
print h.a, h.vec, h.s, h.f(3)
```

5.0 Vector[0] string 9.0

```
h.vec.resize(4)
h.vec.indgen().add(10).printf()
```

10 11 12 13

4.0

Python using Hoc

```
print h.a, h.vec, h.s, h.f(3)
```

5.0 Vector[0] string 9.0

```
h.vec.resize(4)
h.vec.indgen().add(10).printf()
```

10 11 12 13

4.0

```
vx = h.vec.x
print h.vec, vx, vx[2]
```

Vector[0] Vector[0].x[?] 12.0

Python using Hoc

```
import neuron
h = neuron.h
from nrn import *
```

Python using Hoc

```
import neuron
h = neuron.h
from nrn import *

soma = Section()
soma.L = 10
soma(0.5).diam = 10
soma.insert('hh')
```

Python using Hoc

```
import neuron
h = neuron.h
from nrn import *

soma = Section()
soma.L = 10
soma(0.5).diam = 10
soma.insert('hh')

stim = neuron.IClamp(soma, 0.5)
# Thanks Andrew.
stim.delay = 1
stim.dur = 0.2
stim.amp = 0.5
```

Python using Hoc

```
v = neuron.Vector()
soma.push()
v.record(h.ref(soma(0.5).v))
h.pop_section()
```

Python using Hoc

```
v = neuron.Vector()
soma.push()
v.record(h.ref(soma(0.5).v))
h.pop_section()

h.load_file('stdrun.hoc')
h.run()
v.printf()
```

Hoc using Python

Mitral−Granule reciprocal synapse

weight snapshot file
  src tar w

No way to efficiently derive the
  MGRS from the src, tar.

Hoc using Python

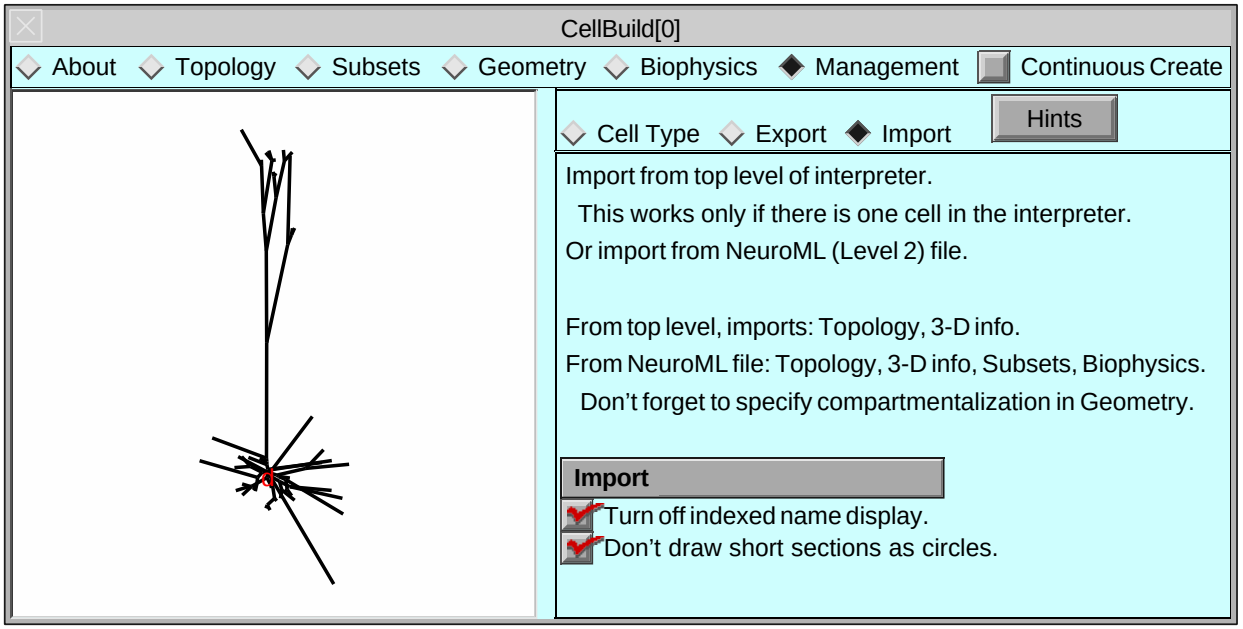
```
objref p, map
p = new PythonObject()
nrnpython(\
  "newmap = lambda key, value : {key:value}")
map = p.newmap(−1,0)
```

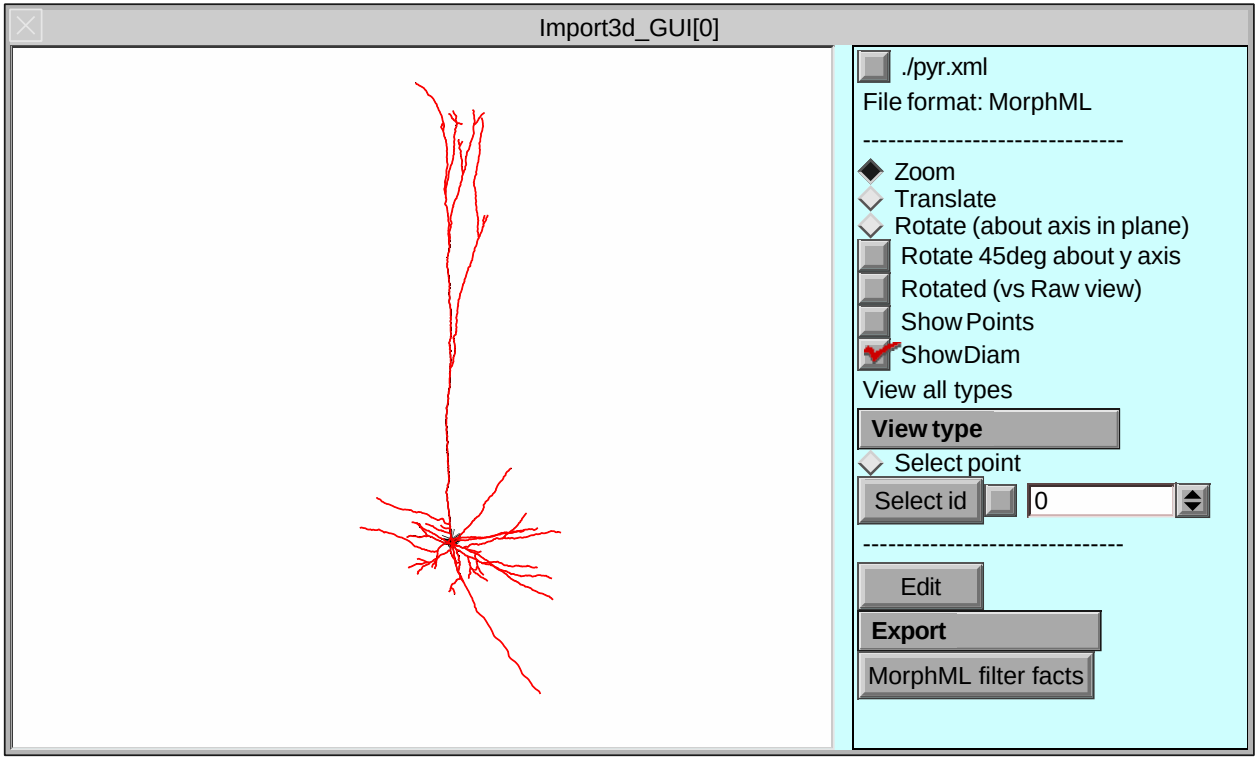
Hoc using Python

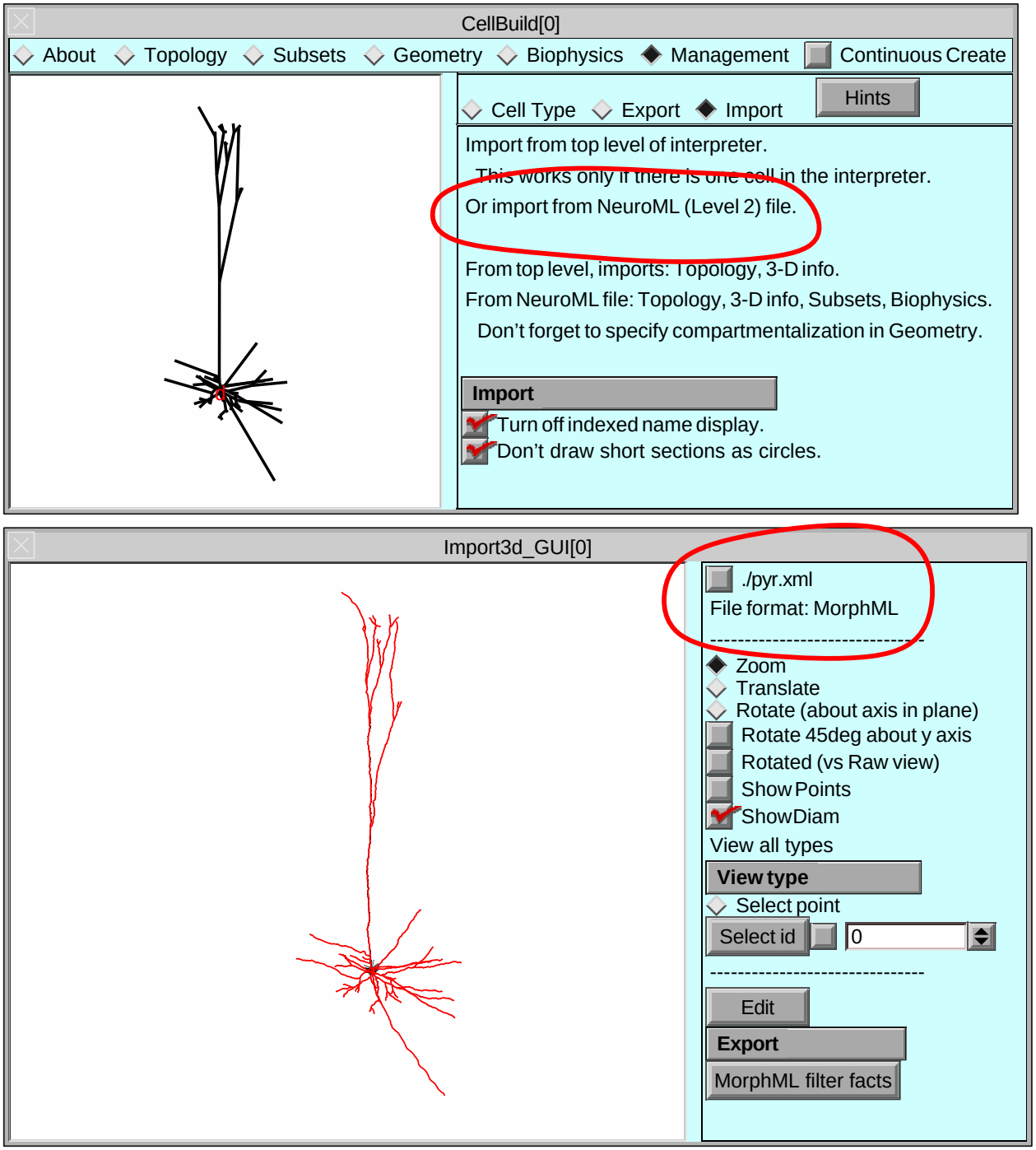
```
objref p, map
p = new PythonObject()
nrnpython(\
  "newmap = lambda key, value : {key:value}")
map = p.newmap(−1,0)

for <lines in file> {
  map.update(p.newmap(gid, w))
}
```

Hoc using Python

```
objref p, map
p = new PythonObject()
nrnpython(\
  "newmap = lambda key, value : {key:value}")
map = p.newmap(−1,0)

for <lines in file> {
  map.update(p.newmap(gid, w))
}

for <all MGRS> {
  w = map._[gid]
}
```


CellBuild[0]
About
Topology
Subsets
Geometry
Biophysics
Management
Continuous Create
Cell Type
Export
Import
Hints
Import from top level of interpreter.
This works only if there is one cell in the interpreter.
Or import from NeuroML (Level 2) file.
From top level, imports: Topology, 3-D info.
From NeuroML file: Topology, 3-D info, Subsets, Biophysics.
Don't forget to specify compartmentalization in Geometry.
Import
Turn off indexed name display.
Don't draw short sections as circles.


Import3d_GUI[0]
./pyr.xml
File format: MorphML
Zoom
Translate
Rotate (about axis in plane)
Rotate 45deg about y axis
Rotated (vs Raw view)
Show Points
ShowDiam
View all types
View type
Select point
Select id
0
Edit
Export
MorphML filter facts


CellBuild[0]
About
Topology
Subsets
Geometry
Biophysics
Management
Continuous Create
Cell Type
Export
Import
Hints
Import from top level of interpreter.
This works only if there is one cell in the interpreter.
Or import from NeuroML (Level 2) file.
From top level, imports: Topology, 3-D info.
From NeuroML file: Topology, 3-D info, Subsets, Biophysics.
Don't forget to specify compartmentalization in Geometry.
Import
Turn off indexed name display.
Don't draw short sections as circles.
Import3d_GUI[0]
./pyr.xml
File format: MorphML
Zoom
Translate
Rotate (about axis in plane)
Rotate 45deg about y axis
Rotated (vs Raw view)
Show Points
ShowDiam
View all types
View type
Select point
Select id
0
Edit
Export
MorphML filter facts

import3d/read_morphml.hoc

```
begintemplate Import3d_MorphML
public parsed
```

import3d/read_morphml.hoc

```
begintemplate Import3d_MorphML
public parsed

proc input() {
  nrnpython("import rdxml")
  p = new PythonObject()
  p.rdxml.__setattr__("i3d", this)
  sprint(tstr, "rdxml.rdxml(’%s’)", $s1)
  nrnpython(tstr)
  p.rdxml.__setattr__("i3d", nil)
}
```

import3d/read_morphml.hoc

```
begintemplate Import3d_MorphML
public parsed

proc input() {
  nrnpython("import rdxml")
  p = new PythonObject()
  p.rdxml.__setattr__("i3d", this)
  sprint(tstr, "rdxml.rdxml(’%s’)", $s1)
  nrnpython(tstr)
  p.rdxml.__setattr__("i3d", nil)
}

proc parsed() { ...
```

python/rdxml.py

```
import xml
i3d = 1

def rdxml(fname) :
  xml.sax.parse(fname, MyContentHandler())
```

python/rdxml.py

```
import xml
i3d = 1

def rdxml(fname) :
  xml.sax.parse(fname, MyContentHandler())

class  Point  Cable  CableGroup  BioParm  BioMech

class MyContentHandler(xml.sax.ContentHandler):

  def endDocument(self):
    i3d.parsed(self)
```

import3d/read_morphml.hoc

```
proc parsed() { ...
  cables = $o1.cables_
  points = $o1.points_
  cableid2index = $o1.cableid2index_
  for i=0, cables.__len__() − 1 {
    cab = cables._[i]
    sec = new Import3d_Section(cab.first_, cab.pcnt_)
    if (cab.parent_cable_id_ >= 0) {
      ip = $o1.cableid2index_[cab.parent_cable_id_]
```