SimPhoNy-Mayavi Documentation

Release 0.1.0.dev0

SimPhoNy FP7 Collaboration

August 04, 2015

Contents

A plugin-library for the Simphony framework (http://www.simphony-project.eu/) to provide visualization support of the CUDS highlevel components.

CHAPTER 1

Repository

Simphony-mayavi is hosted on github: https://github.com/simphony/simphony-mayavi

CHAPTER 2

Requirements

- mayavi >= 4.4.0
- simphony >= 0.1.1

2.1 Optional requirements

To support the documentation built you need the following packages:

- sphinx >= 1.2.3
- sectiondoc commit 8a0c2be, https://github.com/enthought/sectiondoc
- trait-documenter, https://github.com/enthought/trait-documenter
- mock

Alternative running **`pip install -r doc_requirements`** should install the minimum necessary components for the documentation built.

CHAPTER 3

Installation

The package requires python 2.7.x, installation is based on setuptools:

```
# build and install
python setup.py install
```

or:

```
# build for in-place development
python setup.py develop
```

CHAPTER 4

Testing

To run the full test-suite run:

```
python -m unittest discover
```

CHAPTER 5

Documentation

To build the documentation in the doc/build directory run:

```
python setup.py build_sphinx
```

Note:

- One can use the –help option with a setup.py command to see all available options.
- The documentation will be saved in the `./build` directory.

CHAPTER 6

Usage

After installation the user should be able to import the `mayavi` visualization plugin module by:

```
from simphony.visualization import mayavi_tools
mayavi_tools.show(cuds)
```

CHAPTER 7

Directory structure

There are four subpackages:

- simphony-mayavi – Main package code.
- examples – Holds examples of visualizing simphony objects with simphony-mayavi.
- doc – Documentation related files:
  - source – Sphinx rst source files
  - build – Documentation build directory, if documentation has been generated using the `make` script in the `doc` directory.

CHAPTER 8

User Manual

8.1 SimPhoNy

Mayavi tools are available in the simphony library through the visualisation plug-in named `mayavi_tools`.

e.g:

```
from simphony.visualisation import mayavi_tools
```

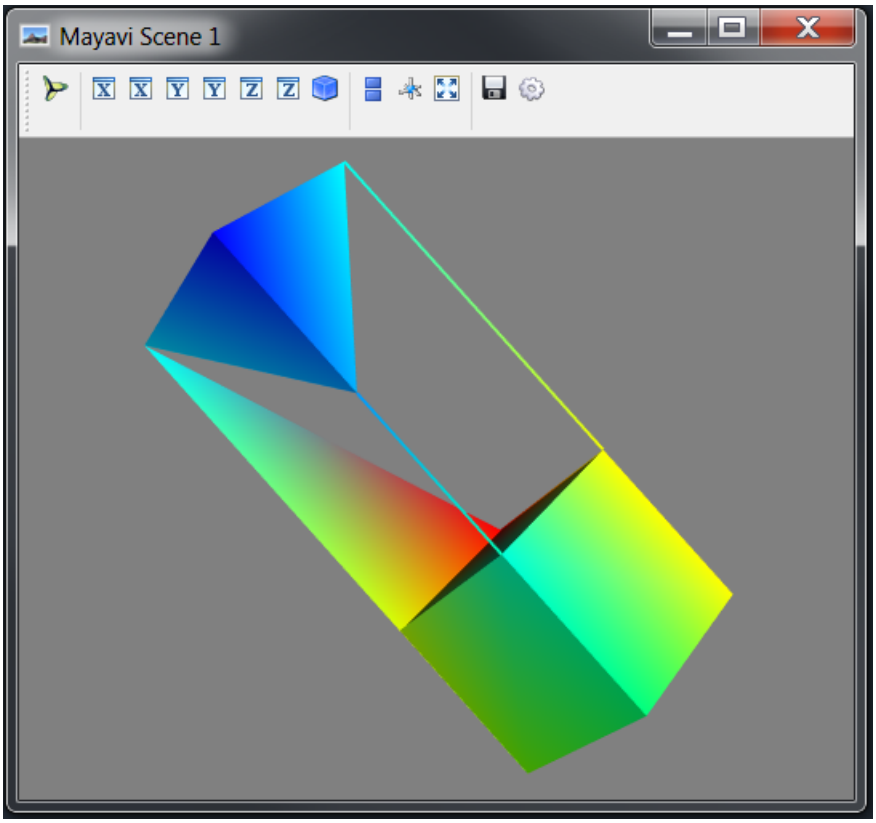
8.1.1 Visualizing CUDS

The *show()* function is available to visualise any top level CUDS container. The function will open a window containing a 3D view and a mayavi toolbar. Interaction allows the common mayavi operations.

Mesh example

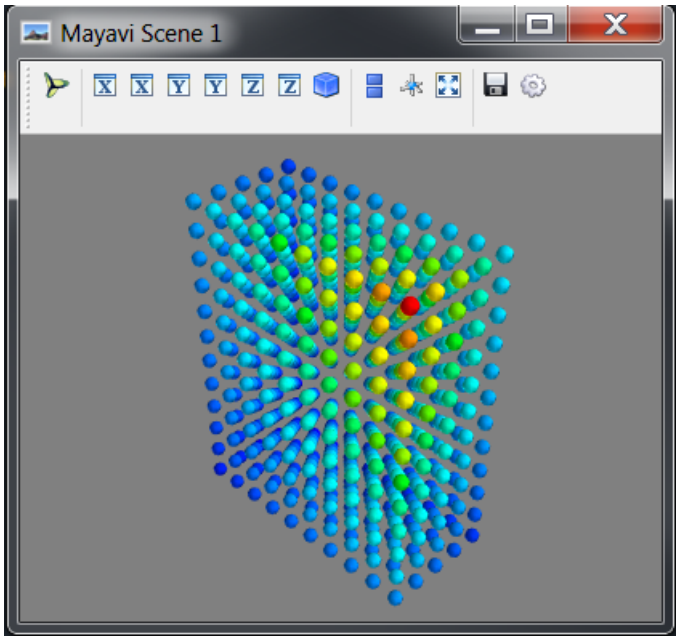
```
from numpy import array

from simphony.cuds.mesh import Mesh, Point, Cell, Edge, Face
from simphony.core.data_container import DataContainer


points = array([
    [0, 0, 0], [1, 0, 0], [0, 1, 0], [0, 0, 1],
    [2, 0, 0], [3, 0, 0], [3, 1, 0], [2, 1, 0],
    [2, 0, 1], [3, 0, 1], [3, 1, 1], [2, 1, 1]],
    'f')

cells = [
    [0, 1, 2, 3],  # tetra
    [4, 5, 6, 7, 8, 9, 10, 11]]  # hex

faces = [[2, 7, 11]]
edges = [[1, 4], [3, 8]]

mesh = Mesh('example')

# add points
uids = [
    mesh.add_point(
```

```
        Point(coordinates=point, data=DataContainer(TEMPERATURE=index)))
    for index, point in enumerate(points)]

# add edges
edge_uids = [
    mesh.add_edge(
        Edge(points=[uids[index] for index in element]))
    for index, element in enumerate(edges)]

# add faces
face_uids = [
    mesh.add_face(
        Face(points=[uids[index] for index in element]))
    for index, element in enumerate(faces)]

# add cells
cell_uids = [
    mesh.add_cell(
        Cell(points=[uids[index] for index in element]))
    for index, element in enumerate(cells)]


if __name__ == '__main__':
    from simphony.visualisation import mayavi_tools

    # Visualise the Mesh object
    mayavi_tools.show(mesh)
```



Lattice example

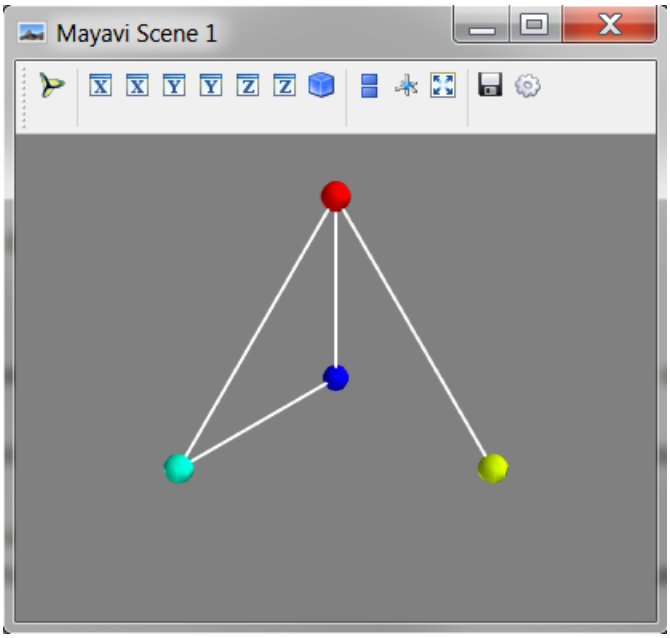
```
import numpy

from simphony.cuds.lattice import make_cubic_lattice
from simphony.core.cuba import CUBA

lattice = make_cubic_lattice('test', 0.1, (5, 10, 12))

for node in lattice.iter_nodes():
    index = numpy.array(node.index) + 1.0
    node.data[CUBA.TEMPERATURE] = numpy.prod(index)
    lattice.update_node(node)


if __name__ == '__main__':
    from simphony.visualisation import mayavi_tools

    # Visualise the Lattice object
    mayavi_tools.show(lattice)
```



Particles example

```
from numpy import array

from simphony.cuds.particles import Particles, Particle, Bond
from simphony.core.data_container import DataContainer

points = array([[0, 0, 0], [1, 0, 0], [0, 1, 0], [0, 0, 1]], 'f')
bonds = array([[0, 1], [0, 3], [1, 3, 2]])
temperature = array([10., 20., 30., 40.])

particles = Particles('test')
```

```
uids = []
for index, point in enumerate(points):
    uid = particles.add_particle(
        Particle(
            coordinates=point,
            data=DataContainer(TEMPERATURE=temperature[index])))
    uids.append(uid)

for indices in bonds:
    particles.add_bond(Bond(particles=[uids[index] for index in indices]))


if __name__ == '__main__':
    from simphony.visualisation import mayavi_tools

    # Visualise the Particles object
    mayavi_tools.show(particles)
```



8.2 Mayavi2

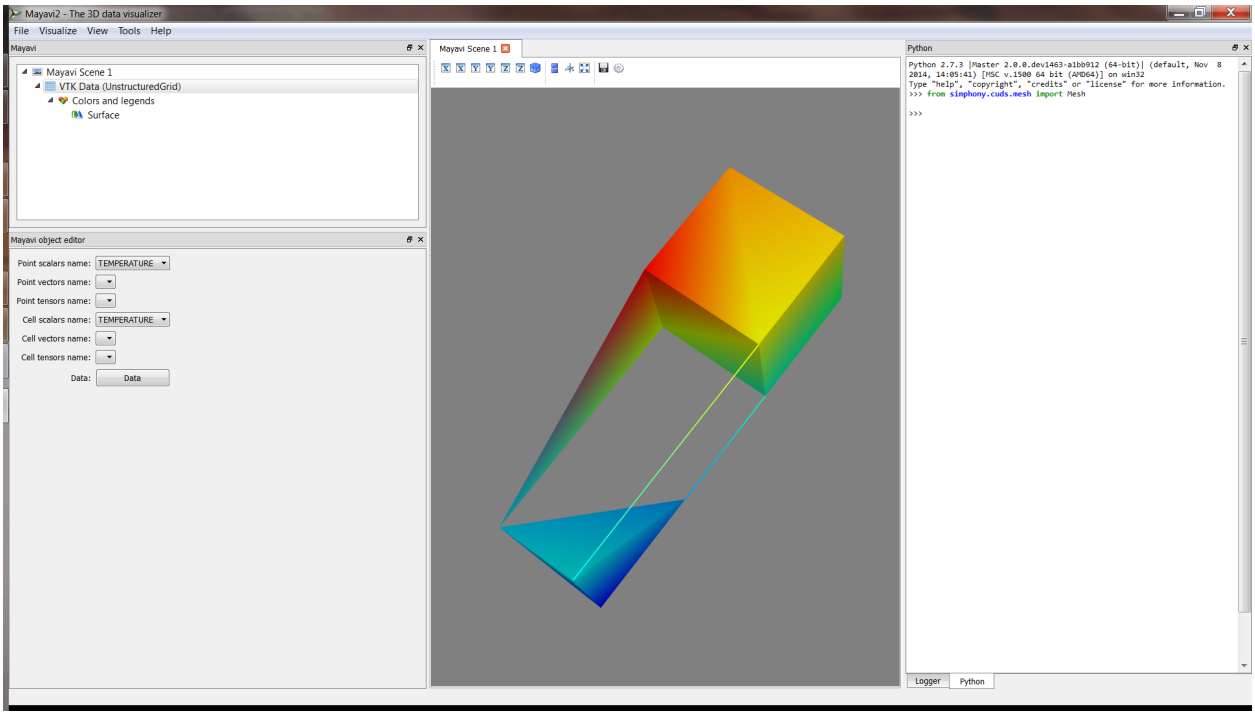
The Simphony-Mayavi library provides a set of tools to easily create mayavi `Source` instances from SimPhoNy CUDS containers. With the provided tools one can use the SimPhoNy libraries to work inside the Mayavi2 application, as it is demonstrated in the examples.

Source from a CUDS Mesh

```
from numpy import array
from mayavi.scripts import mayavi2

from simphony.cuds.mesh import Mesh, Point, Cell, Edge, Face
from simphony.core.data_container import DataContainer
```

```
points = array([
    [0, 0, 0], [1, 0, 0], [0, 1, 0], [0, 0, 1],
    [2, 0, 0], [3, 0, 0], [3, 1, 0], [2, 1, 0],
    [2, 0, 1], [3, 0, 1], [3, 1, 1], [2, 1, 1]],
    'f')

cells = [
    [0, 1, 2, 3],  # tetra
    [4, 5, 6, 7, 8, 9, 10, 11]]  # hex

faces = [[2, 7, 11]]
edges = [[1, 4], [3, 8]]

container = Mesh('test')

# add points
uids = [
    container.add_point(
        Point(coordinates=point, data=DataContainer(TEMPERATURE=index)))
    for index, point in enumerate(points)]

# add edges
edge_uids = [
    container.add_edge(
        Edge(
            points=[uids[index] for index in element],
            data=DataContainer(TEMPERATURE=index + 20)))
    for index, element in enumerate(edges)]

# add faces
face_uids = [
    container.add_face(
        Face(
            points=[uids[index] for index in element],
            data=DataContainer(TEMPERATURE=index + 30)))
    for index, element in enumerate(faces)]

# add cells
cell_uids = [
    container.add_cell(
        Cell(
            points=[uids[index] for index in element],
            data=DataContainer(TEMPERATURE=index + 40)))
    for index, element in enumerate(cells)]


# Now view the data.
@mayavi2.standalone
def view():
    from mayavi.modules.surface import Surface
    from simphony_mayavi.sources.api import MeshSource

    mayavi.new_scene()  # noqa
    src = MeshSource.from_mesh(container)
    mayavi.add_source(src)  # noqa
    s = Surface()
    mayavi.add_module(s)  # noqa
```

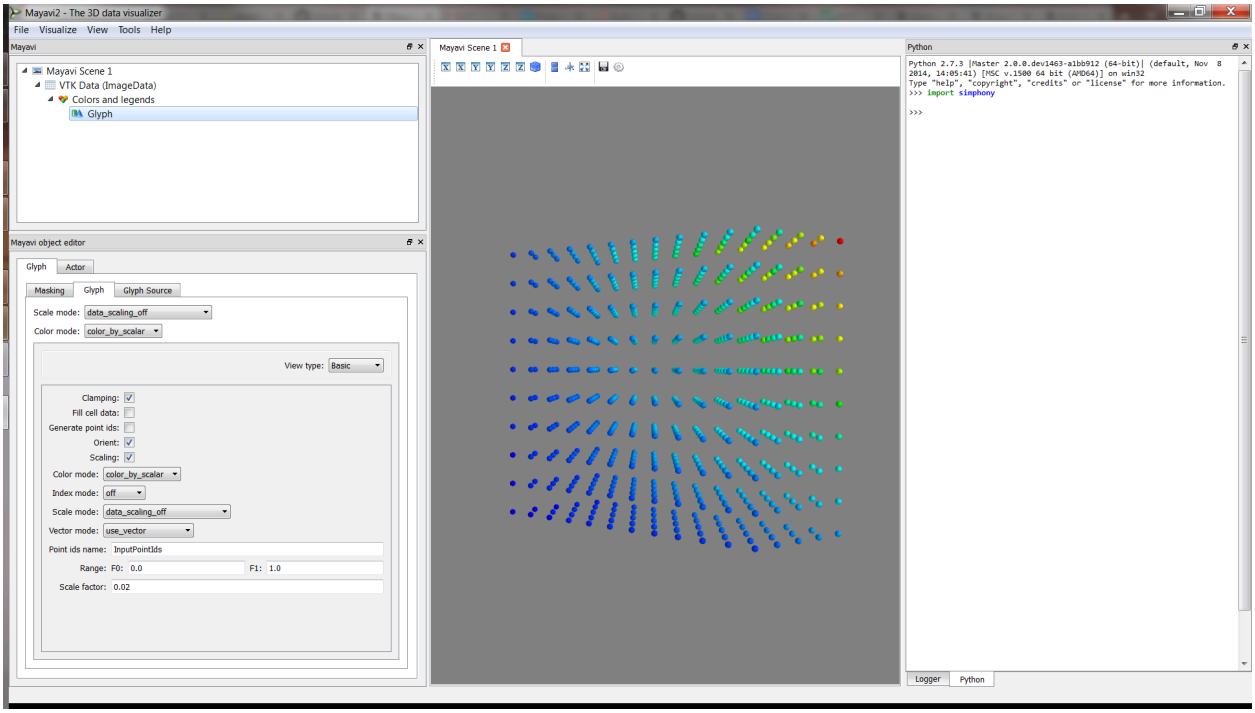
```
if __name__ == '__main__':
    view()
```



Source from a CUDS Lattice

```
import numpy

from mayavi.scripts import mayavi2
from simphony.cuds.lattice import (
    make_hexagonal_lattice, make_cubic_lattice, make_square_lattice)
from simphony.core.cuba import CUBA

hexagonal = make_hexagonal_lattice('test', 0.1, (5, 4))
square = make_square_lattice('test', 0.1, (5, 4))
cubic = make_cubic_lattice('test', 0.1, (5, 10, 12))


def add_temperature(lattice):
    for node in lattice.iter_nodes():
        index = numpy.array(node.index) + 1.0
        node.data[CUBA.TEMPERATURE] = numpy.prod(index)
        lattice.update_node(node)

add_temperature(hexagonal)
add_temperature(cubic)
add_temperature(square)


# Now view the data.
@mayavi2.standalone
```

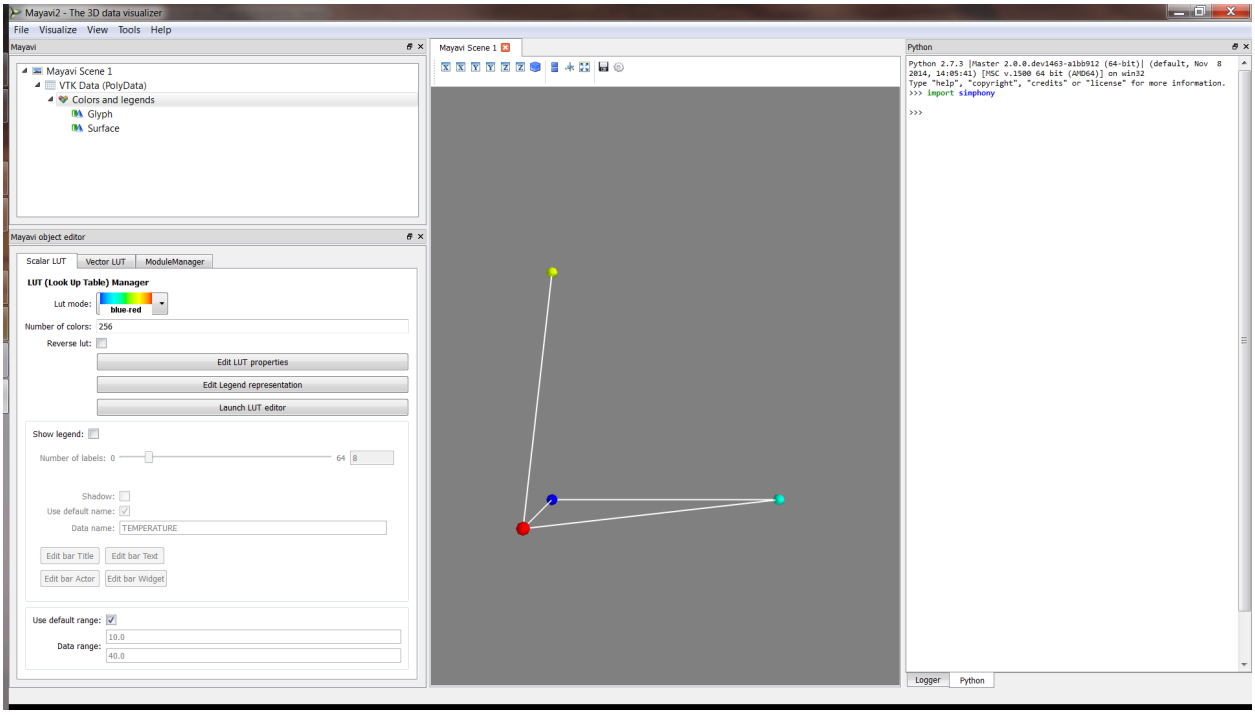
```
def view(lattice):
    from mayavi.modules.glyph import Glyph
    from simphony_mayavi.sources.api import LatticeSource
    mayavi.new_scene()  # noqa
    src = LatticeSource.from_lattice(lattice)
    mayavi.add_source(src)  # noqa
    g = Glyph()
    gs = g.glyph.glyph_source
    gs.glyph_source = gs.glyph_dict['sphere_source']
    g.glyph.glyph.scale_factor = 0.02
    g.glyph.scale_mode = 'data_scaling_off'
    mayavi.add_module(g)  # noqa

if __name__ == '__main__':
    view(cubic)
```



Source for a CUDS Particles

```
from numpy import array
from mayavi.scripts import mayavi2

from simphony.cuds.particles import Particles, Particle, Bond
from simphony.core.data_container import DataContainer

points = array([[0, 0, 0], [1, 0, 0], [0, 1, 0], [0, 0, 1]], 'f')
bonds = array([[0, 1], [0, 3], [1, 3, 2]])
temperature = array([10., 20., 30., 40.])

container = Particles('test')
uids = []
for index, point in enumerate(points):
```

```
    uid = container.add_particle(
        Particle(
            coordinates=point,
            data=DataContainer(TEMPERATURE=temperature[index])))
    uids.append(uid)

for indices in bonds:
    container.add_bond(Bond(particles=[uids[index] for index in indices]))


# Now view the data.
@mayavi2.standalone
def view():
    from mayavi.modules.surface import Surface
    from mayavi.modules.glyph import Glyph
    from simphony_mayavi.sources.api import ParticlesSource

    mayavi.new_scene()  # noqa
    src = ParticlesSource.from_particles(container)
    mayavi.add_source(src)  # noqa
    g = Glyph()
    gs = g.glyph.glyph_source
    gs.glyph_source = gs.glyph_dict['sphere_source']
    g.glyph.glyph.scale_factor = 0.05
    g.glyph.scale_mode = 'data_scaling_off'
    s = Surface()
    s.actor.mapper.scalar_visibility = False

    mayavi.add_module(g)  # noqa
    mayavi.add_module(s)  # noqa

if __name__ == '__main__':
    view()
```



CHAPTER 9

API Reference

9.1 Plugin module

This module `simphony_mayavi.plugin` provides a set of tools to visualize CUDS objects. The tools are also available as a visualisation plug-in to the simphony library.

`simphony_mayavi.show.`**`show`**(*cuds*)
Show the cuds objects using the default visualisation.

Parameters **`cuds`** – A top level cuds object (e.g. a mesh). The method will detect the type of object and create the appropriate visualisation.

`simphony_mayavi.snapshot.`**`snapshot`**(*cuds*, *filename*)
Shave a snapshot of the cuds object using the default visualisation.

Parameters

- **`cuds`** – A top level cuds object (e.g. a mesh). The method will detect the type of object and create the appropriate visualisation.
- **`filename`** (*string*) – The filename to use for the output file.

9.2 Sources module

A module containing tools to convert from CUDS containers to Mayavi compatible sources. Please use the `simphony_mayavi.sources.api` module to access the provided tools.

Classes

ParticlesSource	SimPhoNy CUDS Particle container to Mayavi Source converter
LatticeSource	SimPhoNy CUDS Lattice container to Mayavi Source converter
MeshSource	SimPhoNy CUDS Mesh container to Mayavi Source converter
CUDSDataAccumulator([keys])	Accumulate data information per CUBA key.
CUDSDataExtractor(**traits)	Extract data from cuds items iterable.

Functions

cell_array_slicer(data)	Iterate over cell components on a vtk cell array

9.2.1 Description

class simphony_mayavi.sources.particles_source.**ParticlesSource**
Bases: mayavi.sources.vtk_data_source.VTKDataSource

SimPhoNy CUDS Particle container to Mayavi Source converter

bond2index = Dict
The mapping from the bond uid to the vtk polydata cell index.

classmethod from_particles(*particles*)
Return a ParticlesSource from a CUDS Particles container.

Parameters **particles** (*Particles*) – The CUDS Particles instance to copy the information from.

point2index = Dict
The mapping from the point uid to the vtk polydata points array.

class simphony_mayavi.sources.mesh_source.**MeshSource**
Bases: mayavi.sources.vtk_data_source.VTKDataSource

SimPhoNy CUDS Mesh container to Mayavi Source converter

element2index = Dict
The mapping from the element uid to the vtk cell index.

classmethod from_mesh(*mesh*)
Return a MeshSource from a CUDS Mesh container.

Parameters **mesh** (*Mesh*) – The CUDS Mesh instance to copy the information from.

point2index = Dict
The mapping from the point uid to the vtk points array.

class simphony_mayavi.sources.lattice_source.**LatticeSource**
Bases: mayavi.sources.vtk_data_source.VTKDataSource

SimPhoNy CUDS Lattice container to Mayavi Source converter

classmethod from_lattice(*lattice*)
Return a LatticeSource from a CUDS Lattice container.

Parameters **lattice** (*Lattice*) – The cuds Lattice instance to copy the information from.

class simphony_mayavi.sources.cuds_data_accumulator.**CUDSDataAccumulator**(*keys=()*)
Bases: object

Accumulate data information per CUBA key.

A collector object that stores :class:DataContainer data into a list of values per CUBA key. By appending DataContainer instanced the user can effectively convert the per item mapping of data values in a CUDS container to a per CUBA key mapping of the data values (useful for coping data to vtk array containers).

The Accumulator has two modes of operation fixed and expand. fixed means that data will be stored for a predefined set of keys on every append call and missing values will be saved as None. Where expand will extend the internal table of values when ever a new key is introduced.

expand operation

```
>>> accumulator = CUDSDataAccumulator():
>>> accumulator.append(DataContainer(TEMPERATURE=34))
>>> accumulator.keys()
{CUBA.TEMPERATURE}
>>> accumulator.append(DataContainer(VELOCITY=(0.1, 0.1, 0.1))
>>> accumulator.append(DataContainer(TEMPERATURE=56))
>>> accumulator.keys()
{CUBA.TEMPERATURE, CUBA.VELOCITY}
>>> accumulator[CUBA.TEMPERATURE]
[34, None, 56]
>>> accumulator[CUBA.VELOCITY]
[None, (0.1, 0.1, 0.1), None]
```

fixed operation

```
>>> accumulator = CUDSDataAccumulator([CUBA.TEMPERATURE, CUBA.PRESSURE]):
>>> accumulator.keys()
{CUBA.TEMPERATURE, CUBA.PRESSURE}
>>> accumulator.append(DataContainer(TEMPERATURE=34))
>>> accumulator.append(DataContainer(VELOCITY=(0.1, 0.1, 0.1))
>>> accumulator.append(DataContainer(TEMPERATURE=56))
>>> accumulator.keys()
{CUBA.TEMPERATURE, CUBA.PRESSURE}
>>> accumulator[CUBA.TEMPERATURE]
[34, None, 56]
>>> accumulator[CUBA.PRESSURE]
[None, None, None]
>>> accumulator[CUBA.VELOCITY]
KeyError(...)
```

Constructor

Parameters **keys** (*list*) – The list of keys that the accumulator should care about. Providing this value at initialisation sets up the accumulator to operate in `fixed` mode. If no keys are provided then accumulator operates in `expand` mode.

__getitem__(*key*)
Get the list of accumulated values for the CUBA key.

Parameters **key** (*CUBA*) – A CUBA Enum value

Returns **result** (*list*) – A list of data values collected for `key`. Missing values are designated with `None`.

__len__()
The number of values that are stored per key

Note: Behaviour is temporary and will probably change soon.

append(*data*)
Append info from a `DataContainer`.

Parameters **data** (*DataContainer*) – The data information to append.

If the accumulator operates in `fixed` mode:

- Any keys in `self.keys()` that have values in `data` will be stored (appended to the related key lits).
- Missing keys will be stored as `None`

If the accumulator operates in `expand` mode:

- Any new keys in *Data* will be added to the `self.keys()` list and the related list of values with length equal to the current record size will be initialised with values of `None`.
- Any keys in the modified `self.keys()` that have values in `data` will be stored (appended to the list of the related key).
- Missing keys will be store as `None`.

keys
The set of CUBA keys that this accumulator contains.

load_onto_vtk(*vtk_data*)
Load the stored information onto a vtk data container.

Parameters **vtk_data** (*vtkPointData or vtkCellData*) – The vtk container to load the value onto.

Data are loaded onto the vtk container based on their data type. The name of the added array is the name of the CUBA key (i.e. *CUBA*.name). Currently only scalars and three dimensional vectors are supported.

class simphony_mayavi.sources.cuds_data_extractor.**CUDSDataExtractor**(***traits*)
Bases: traits.has_traits.HasStrictTraits

Extract data from cuds items iterable.

The class that supports extracting data values of a specific CUBA key from an iterable that returns low level CUDS objects (e.g. `Point`).

available = Property(Set(CUBATrait), depends_on='_available')
The list of cuba keys that are available (read only). The value is recalculated at initialisation and when the `reset` method is called.

data = Property(Dict(UUID, Any), depends_on='_data')
The dictionary mapping of item uid to the extracted data value. A change Event is fired for `data` when `selected` or `keys` change or the `reset` method is called.

function = ReadOnly
The function to call that returns a generator over the desired items (e.g. Mesh.iter_points). This value cannot be changed after initialisation.

keys = Either(None, Set(UUID))
The list of uuid keys to restrict the data extraction. This attribute is passed to the function generator method to restrict iteration over the provided keys (e.g Mesh.iter_points(uids=keys))

reset()
Reset the `available` and `data` attributes.

selected = CUBATrait
Currently selected CUBA key. Changing the selected key will fire events that will result in executing the generator function and extracting the related values from the CUDS items that the iterator yields. The resulting mapping of `uid -> value` will be stored in `data`.

simphony_mayavi.sources.utils.**cell_array_slicer**(*data*)
Iterate over cell components on a vtk cell array

VTK stores the associated point index for each cell in a one dimensional array based on the following template:

```
[n, id0, id1, id2, ..., idn, m, id0, ...]
```

The iterator takes a cell array and returns the point indices for each cell.

CHAPTER 10

Simphony-Mayavi

A plugin-library for the Simphony framework (http://www.simphony-project.eu/) to provide visualization support of the CUDS highlevel components.

10.1 Repository

Simphony-mayavi is hosted on github: https://github.com/simphony/simphony-mayavi

10.2 Requirements

- mayavi >= 4.4.0
- simphony >= 0.1.1

10.2.1 Optional requirements

To support the documentation built you need the following packages:

- sphinx >= 1.2.3
- sectiondoc commit 8a0c2be, https://github.com/enthought/sectiondoc
- trait-documenter, https://github.com/enthought/trait-documenter
- mock

Alternative running **`pip install -r doc_requirements`** should install the minimum necessary components for the documentation built.

10.3 Installation

The package requires python 2.7.x, installation is based on setuptools:

```
# build and install
python setup.py install
```

or:

```
# build for in-place development
python setup.py develop
```

10.4 Testing

To run the full test-suite run:

```
python -m unittest discover
```

10.5 Documentation

To build the documentation in the doc/build directory run:

```
python setup.py build_sphinx
```

Note:

- One can use the –help option with a setup.py command to see all available options.
- The documentation will be saved in the `./build` directory.

10.6 Usage

After installation the user should be able to import the `mayavi` visualization plugin module by:

```
from simphony.visualization import mayavi_tools
mayavi_tools.show(cuds)
```

10.7 Directory structure

There are four subpackages:

- simphony-mayavi – Main package code.
- examples – Holds examples of visualizing simphony objects with simphony-mayavi.
- doc – Documentation related files:
  - source – Sphinx rst source files
  - build – Documentation build directory, if documentation has been generated using the `make` script in the `doc` directory.

Symbols

A

B

C

D

E

F

K

L

M

P

R

S