# **Cortix Documentation**

Release 0.1.0

Valmor F. de Almeida, Taha Azzaoui, Gilberto E. Alas

# **CONTENTS**

1	src		]
	1.1	cortix_main	
	1.2	module module	2
	1.3	network module	2
	1.4	port module	4
2	exam	ples	-
	2.1	body module	7
	2.2	dataplot module	8
	2.3	dummy_module module	8
	2.4	plot_data module	Ç
	2.5	run_planets module	10
	2.6	run_region_justice module	10
	2.7	state module	1.
3		et_swirl	13
	3.1	droplet module	13
	3.2	run_droplet_swirl module	14
	3.3	run_droplet_test module	15
	3.4	vortex module	15
			4.
4	city_	ustice	17
4	city_; 4.1	ustice adjudication module	17 17
4			
4	4.1	adjudication module	17
4	4.1 4.2	adjudication module	17 18
4	4.1 4.2 4.3	adjudication module	13 18 19
4	4.1 4.2 4.3 4.4	adjudication module	13 18 19 20
4	4.1 4.2 4.3 4.4 4.5	adjudication module arrested module community module jail module parole module parole module community modul	13 18 19 20 21
4	4.1 4.2 4.3 4.4 4.5 4.6	adjudication module arrested module community module jail module parole module prison module	13 18 19 20 21 22
	4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8	adjudication module arrested module community module jail module parole module prison module probation module probation module run_city_justice module	13 18 19 20 21 22 23 24
5	4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8	adjudication module arrested module community module jail module parole module prison module probation module run_city_justice module	13 18 19 20 21 22 23 24 25
	4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 <b>supp</b> 5.1	adjudication module arrested module community module jail module parole module prison module probation module run_city_justice module priodictable module	13 18 19 20 21 22 23 24 25
	4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 <b>supp</b> 5.1 5.2	adjudication module arrested module community module jail module parole module prison module probation module run_city_justice module  periodictable module phase module.	17. 18. 19. 20. 21. 22. 23. 24. 25. 26.
	4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 supposes	adjudication module arrested module community module jail module parole module prison module probation module run_city_justice module  periodictable module phase module quantity module  arrested module parole module parole module prison module probation module probation module probation module arrested module prison module probation module arrested module prison module arrested module prison module arrested mod	17.18 19.20 21.22 22.22 24.22 26.30
	4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 supposes 5.1 5.2 5.3 5.4	adjudication module arrested module community module jail module parole module prison module probation module run_city_justice module  periodictable module phase module quantity module specie module	17.18 19.20 20.22 20.22 20.22 20.30 30.33
	4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 supposes	adjudication module arrested module community module jail module parole module prison module probation module run_city_justice module  periodictable module phase module quantity module  arrested module parole module parole module prison module probation module probation module probation module arrested module prison module probation module arrested module prison module arrested module prison module arrested mod	17.18 19.20 21.22 22.22 24.22 26.30
5	4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 supposes 5.1 5.2 5.3 5.4	adjudication module arrested module community module jail module parole module prison module probation module run_city_justice module  prison module prison module run_city_justice module  prison module prison module arrested module prison module probation module probation module  arrested module prison module prison module probation module prison module prison module arrested module prison module prison module prison module arrested module prison module arrested module prison module arrested module prison module arrested	17.18 19.20 20.22 20.22 20.22 20.30 30.33
5	4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 suppo 5.1 5.2 5.3 5.4 5.5	adjudication module arrested module community module jail module parole module prison module probation module run_city_justice module  periodictable module phase module quantity module specie module stream module stream module ar actor module	10 18 19 20 20 20 20 20 20 20 20 20 20 20 20 20
5	4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 suppo 5.1 5.2 5.3 5.4 5.5	adjudication module arrested module community module jail module parole module prison module probation module run_city_justice module  prison module prison module run_city_justice module  prison module prison module arrested module prison module probation module probation module  arrested module prison module prison module probation module prison module prison module arrested module prison module prison module prison module arrested module prison module arrested module prison module arrested module prison module arrested	10 18 19 20 20 22 22 22 24 25 26 30 31 38

Index				
Python Module Index				
6.7	nuclides module	56		
6.6	fuelslug module	55		
	fuelsegmentsgroups module			
6.4	fuel_segment module	52		
6.3	fuel_bundle module	4		

**CHAPTER** 

ONE

**SRC** 

# 1.1 cortix\_main

class cortix\_main.Cortix (use\_mpi=False, splash=False)
 Bases: object

Cortix main class definition.

The typical Cortix run file workflow:

- 1. Create the *Cortix* object
- 2. Create tne (nested) network of modules
- 3. Run and close Cortix
- del ()

Destructs a Cortix simulation object.

**Warning:** By the time the body of this function is executed, the machinery of variables may have been deleted already. For example, *logging* is no longer there; do the least amount of work here.

```
__init__ (use_mpi=False, splash=False)
Construct a Cortix simulation object.
```

### **Parameters**

- use\_mpi (bool) True for MPI, False for multiprocessing.
- **splash** (bool) Show the Cortix splash image.

### network

Network - A network of modules and their connectivity.

### use\_mpi

bool - True for MPI, False for Multiprocessing.

### use\_multiprocessing

bool - False for MPI, True for Multiprocessing.

### splash

bool - Show the Cortix splash image.

### comm

mpi4py.MPI.Intracomm - MPI.COMM\_WORLD (if using MPI else None).

#### rank

int – The current MPI rank (if using MPI else None).

#### size

int – size of the group associated with MPI.COMM\_WORLD.

#### close()

Closes the cortix object properly before destruction.

User is strongly advised to call this method at the end of the run file otherwise timings will not be recorded.

### network

#### run()

Run the Cortix network simulation.

# 1.2 module module

### class module. Module

Bases: object

Cortix module super class.

This class provides facilities for creating modules within the Cortix network. Cortix will map one object of this class to either a Multiprocessing or MPI process depending on the user's configuration.

**Note:** This class is to be inherited by every Cortix module. In order to execute, modules *must* override the *run* method, which will be executed during the simulation.

```
init ()
```

Module super class constructor.

**Note:** This constructor must be called explicitly in the constructor of every Cortix module like so:

```
super().__init__()
```

### name

str – A name given to the instance. Default is the derived class name.

### port\_names\_expected

*list(str)*, *None* – A list of names of ports expected in the module. This will be compared to port names during runtime to check against the intended use of the module.

### state

any – Any pickle-able data structure to be passed in a multiprocessing. Queue to the parent process or to be gathered in the root MPI process. Default is None.

### use\_mpi

bool - True for MPI, False for Multiprocessing

# use\_multiprocessing

bool - False for MPI, True for Multiprocessing

### ports

list(Port) – A list of ports contained by the module

1.2. module module 2

```
network
```

*Network* – An internal network inherited by the derived module for nested networks.

### get\_port (name)

Get port by name; if it does not exist, create one.

**Parameters** name (str) – The name of the port to get

**Returns port** – The port object with the corresponding name

Return type Port

### network

recv (port)

Receive data from a given port

Warning: This function will block until data is available

Parameters port (Port, str) - A Port object to send the data through, or its string name

**Returns data** – The data received through the port

Return type any

run (\*args)

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### **Notes**

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use multiprocessing:

try: pickle.dumps(self.state)

**except pickle.PicklingError:** args[1].put((arg[0],None))

**else:** args[1].put((arg[0],self.state))

at the bottom of the user defined *run()* function.

Warning: This function must be overridden by all Cortix modules

### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

1.2. module module 3

```
send (data, port)
```

Send data through a given port.

### **Parameters**

- data (any) The data being sent out must be pickleable
- port (Port, str) A Port object to send the data through, or its string name

# 1.3 network module

### class network.Network

Bases: object

Cortix network.

#### n networks

int – Number of instances of this class.

```
___init___()
```

Module super class constructor.

### max\_n\_modules\_for\_data\_copy\_on\_root

*int* – When using MPI the *network* will copy the data from all modules on the root process. This can generate an *out of memory* condition. This variable sets the maximum number of processes for which the data will be copied. Default is 1000.

### add module (m)

Alternative name to *module()*.

```
connect (module port a, module port b, info=None)
```

Connect two modules using either their ports directly or inferred ports.

A connection always opens a channel for data communication in both ways. That is, both sends and receives are allowed.

**Note:** The simplest form of usage is with arguments:  $(module\_a, module\_b)$ . In this case, a *port* with the name of  $module\_a$  **must** exist in  $module\_b$ , and vice-versa (port names as str in lower case). In addition, the connect must not be called again with these same two modules, else the underlying connection will be overriden.

For more rigorous connection, the user is advised to fully specify the module and the port in each list argument.

### **Parameters**

- module\_port\_a (list([Module,Port]) or list([Module,str]) or Module) First module-port to connect.
- module\_port\_b (list([Module,Port]) or list([Module,str]) or Module) Second module-port to connect.
- **info** (str) Information on the directionality of the information flow. This is for graph visualization purposes only. The default value will use the order in the argument list to define the direction. Default: None. If set to *bidiretional*, will create a double headed arrow in the graph figure.

draw (graph\_attr=None, node\_attr=None, engine=None)

1.3. network module 4

```
module (m)
    Add a module.

n_networks = 0
run ()
    Run the network simulation.
```

This function concurrently executes the *cortix.src.module.run* function for each module in the network. Modules are run using either MPI or Multiprocessing, depending on the user configuration.

**Note:** When using multiprocessing, data from the modules state are copied to the master process after the *run()* method of the modules is finished.

# 1.4 port module

```
class port.Port (name=None, use_mpi=False)
    Bases: object
```

Provides a method of communication between modules.

The Port class provides an interface for creating ports and connecting them to other ports for the purpose of data transfer. Data exchange takes place by send and/or receive calls on a given port. The concept of a port is that of a data transfer "interaction." This can be one- or two-way with sends and receives. A port is connected to only one other port; as two ends of a pipe are connected.

```
\underline{\phantom{a}}eq\underline{\phantom{a}} (other)
     Check for port equality
__init__ (name=None, use_mpi=False)
     Constructs a Port object
          Parameters
               • name (str) - The name of the Port object
               • use_mpi (bool) - True for MPI, False for Multiprocessing
     id
          int
     name
          string
     use_mpi
          bool
__repr__()
     Port name representation
connect (port)
     Connect this port to another port
     Ports must be connected for data to flow between them.
          Parameters port (Port) – A Port object to connect to
recv()
     Receive data from the connected port.
```

1.4. port module 5

Warning: This function will block if no data has been sent yet.

# Returns data

Return type any

send(data, tag=None)

Send data to the connected port.

If the sending port is not connected do nothing.

### **Parameters**

- data (any) This data must be pickleable
- tag (int, optional) MPI tag used in sending data

1.4. port module 6

**CHAPTER** 

**TWO** 

# **EXAMPLES**

# 2.1 body module

```
class body.Body (mass=0, rad=0, pos=(0, 0, 0), vel=(0, 0, 0))
    Bases: cortix.src.module.Module
    acceleration (body)
    run()
        Module run function
```

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

# Notes

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

```
try: pickle.dumps(self.state)
    except pickle.PicklingError: args[1].put((arg[0],None))
    else: args[1].put((arg[0],self.state))
at the bottom of the user defined run() function.
```

Warning: This function must be overridden by all Cortix modules

### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 2.2 dataplot module

```
class dataplot.DataPlot
   Bases: cortix.src.module.Module

plot_data()

recv_data(port)

   Keep listening on the port and receiving data.

run(*args)

   Spawn a thread to handle each port connection.
```

# 2.3 dummy module module

```
class dummy_module.DummyModule
    Bases: cortix.src.module.Module
    run()
        Module run function
```

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### **Notes**

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

```
try: pickle.dumps(self.state)
    except pickle.PicklingError: args[1].put((arg[0],None))
    else: args[1].put((arg[0],self.state))
at the bottom of the user defined run() function.
```

Warning: This function must be overridden by all Cortix modules

### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm, self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

```
class dummy_module.DummyModule2
    Bases: cortix.src.module.Module
```

run()

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

#### Notes

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

```
try: pickle.dumps(self.state)
```

except pickle.PicklingError: args[1].put((arg[0],None))

**else:** args[1].put((arg[0],self.state))

at the bottom of the user defined *run()* function.

Warning: This function must be overridden by all Cortix modules

#### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 2.4 plot\_data module

```
class plot_data.PlotData
   Bases: cortix.src.module.Module
```

run()

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### Notes

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

try: pickle.dumps(self.state)

```
except pickle.PicklingError: args[1].put((arg[0],None))
```

**else:** args[1].put((arg[0],self.state))

at the bottom of the user defined run() function.

Warning: This function must be overridden by all Cortix modules

### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 2.5 run\_planets module

```
run_planets.main()
```

# 2.6 run\_region\_justice module

Crimninal justice network dynamics modeling.

This example uses 7 modules:

- Community
- Arrested
- Adjudication
- Jail
- Prison
- Probation
- Parole

and a population balance model is used to follow the offenders population groups between modules.

To run this case using MPI you should compute the number of processes as follows:

```
nprocs = 7 + 1 cortix
```

then issue the MPI run command as follows (replace *nprocs* with a number):

```
mpiexec -n nprocs run_justice.py
```

To run this case with the Python multiprocessing library, just run this file at the command line as

```
run_justice.py
```

```
run_region_justice.main()
```

Cortix run file for a criminal justice network.

```
run_region_justice.n_groups
    int – Number of population groups being followed. This must be the same for all modules.
run_region_justice.end_time
    float – End of the flow time in SI unit.
run region justice.time step
    float – Size of the time step between port communications in SI unit.
run region justice.use mpi
    bool - If set to True use MPI otherwise use Python multiprocessing.
```

# 2.7 state module

```
class state.State (name, non_offender_adult_population=100)
    Bases: cortix.src.module.Module
```

State Cortix module used to model non-offender group population transit from and to a state. This assumes various ports of communication with other states, and an internal port to the internal Community.

### **Notes**

These are the port names available in this module to connect to other State modules: inflow:id, outflow:id. In addition this module takes an internal network to model the free-offenders community of people. The port used for this connection is *community*. See instance attribute *port\_names\_expected*.

```
__init__ (name, non_offender_adult_population=100)
       Parameters non_offender_adult_population (float) - Individuals reaching the
           adult age (SI) unit. Default: 100.
```

Module run function

run (\*args)

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### Notes

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index *comm\_idx* and the *state* into the queue as follows:

### if self.use\_multiprocessing:

```
try: pickle.dumps(self.state)
        except pickle.PicklingError: args[1].put((arg[0],None))
        else: args[1].put((arg[0],self.state))
at the bottom of the user defined run() function.
```

Warning: This function must be overridden by all Cortix modules

2.7. state module 11

### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

2.7. state module

**CHAPTER** 

# **THREE**

# DROPLET\_SWIRL

# 3.1 droplet module

```
class droplet.Droplet
    Bases: cortix.src.module.Module
```

Droplet Cortix module used to model very simple fluid-particle interactions.

### Notes

Port names used in this module: *external-flow* exchanges data with any other module that provides information about the flow outside the droplet, *visualization* sends data to a visualization module.

```
__init__()

initial_time
    float

end_time
    float

time_step
    float

show_time
    tuple - Two-element tuple, (bool,float), True will print to standard output.

run (*args)
```

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### **Notes**

Module run function

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

```
try: pickle.dumps(self.state)
except pickle.PicklingError: args[1].put((arg[0],None))
```

```
else: args[1].put((arg[0],self.state))
```

at the bottom of the user defined *run()* function.

Warning: This function must be overridden by all Cortix modules

#### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state comm must have the module's self.state in it. state comm.put((idx comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 3.2 run droplet swirl module

This example uses two modules instantiated many times. It be executed with MPI (if mpi4py is available) or with the Python multiprocessing library. These choices are made by variables listed below in the executable portion of this run

To run this case using MPI you should compute the number of processes as follows:

```
nprocs = n\_droplets + 1 \ vortex + 1 \ cortix
```

then issue the MPI run command as follows (replace *nprocs* with a number):

```
mpiexec -n nprocs run_droplet.py
```

To run this case with the Python multiprocessing library, just run this file at the command line as

```
run_droplet.py
run_droplet_swirl.main()
     Cortix run file for a Droplet-Vortex network.
     run_droplet_swirl.n_droplets
          int – Number of droplets to use (one per process).
     run_droplet_swirl.end_time
         float – End of the flow time in SI unit.
     run_droplet_swirl.time_step
         float – Size of the time step between port communications in SI unit.
     run_droplet_swirl.create_plots
          bool - Create various plots and save to files. (all data collected in the parent process; it may run out of
          memory).
     run_droplet_swirl.plot_vortex_profile
```

bool – Whether to plot (to a file) the vortex function used.

```
run_droplet_swirl.use_mpi
```

bool – If set to True use MPI otherwise use Python multiprocessing.

# 3.3 run\_droplet\_test module

This example uses three modules instantiated many times in two different networks. Each network configuration uses a different amount of module instances and a different network topology. This example can be executed with MPI (if mpi4py is available) or with the Python multiprocessing library. These choices are made by variables listed below in the executable portion of this run file.

# 3.3.1 Single Plot

The first network case is named "single plot". Here one DataPlot module is connected to all Droplet modules. To run this case using MPI you should compute the number of processes as follows:

```
nprocs = n\_droplets + 1 \ vortex + 1 \ data\_plot + 1 \ cortix
```

then issue the MPI run command as follows (replace *nprocs* with a number):

```
mpiexec -n nprocs run_droplet.py
```

To run this case with the Python multiprocessing library, just run this file at the command line as

```
run_droplet.py
```

# 3.3.2 Multiple Plot

The second network case is named "multiple plot". Here each Droplet is connected to an instance of the DataPlot module, therefore many more nodes are added to the network when compared to the first network case. To run this case using MPI compute

```
nprocs = 2*n\_droplets + 1 \ vortex + 1 \ cortix
```

then issue the MPI run command as follows (replace nprocs:

```
mpiexec -n nprocs run_droplet.py
```

To run this case with the Python multiprocessing library, just run this file at the command line as

```
run_droplet.py
```

# 3.4 vortex module

```
class vortex.Vortex
```

```
Bases: cortix.src.module.Module
```

Vortex module used to model fluid flow using Cortix.

### Notes

Any port name and any number of ports are allowed.

```
__init__()

initial_time

float
```

plot\_velocity(time=None)

Plot the vortex velocity as a function of height.

run (\*args)

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### Notes

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

```
try: pickle.dumps(self.state)
except pickle.PicklingError: args[1].put((arg[0],None))
else: args[1].put((arg[0],self.state))
```

at the bottom of the user defined *run()* function.

Warning: This function must be overridden by all Cortix modules

### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

3.4. vortex module

**CHAPTER** 

# **FOUR**

# CITY JUSTICE

# 4.1 adjudication module

```
class adjudication.Adjudication(n_groups=1, pool_size=0.0)
    Bases: cortix.src.module
```

Adjudication Cortix module used to model criminal group population in an adjudication system.

#### **Notes**

These are the *port* names available in this module to connect to respective modules: *probation*, *jail*, *arrested*, *prison*, and *community*. See instance attribute *port\_names\_expected*.

```
___init___(n_groups=1, pool_size=0.0)
```

### **Parameters**

- n\_groups (int) Number of groups in the population.
- **pool\_size** (float) Upperbound on the range of the existing population groups. A random value from 0 to the upperbound value will be assigned to each group.

run (\*args)

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### **Notes**

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

at the bottom of the user defined *run()* function.

```
try: pickle.dumps(self.state)
except pickle.PicklingError: args[1].put((arg[0],None))
else: args[1].put((arg[0],self.state))
```

Warning: This function must be overridden by all Cortix modules

### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 4.2 arrested module

```
class arrested.Arrested(n_groups=1, pool_size=0.0)
Bases: cortix.src.module.Module
```

Arrested Cortix module used to model criminal group population in an arrested system.

#### **Notes**

These are the *port* names available in this module to connect to respective modules: *probation*, *adjudication*, *jail*, and *community*. See instance attribute *port\_names\_expected*.

```
___init__ (n_groups=1, pool_size=0.0)
```

### **Parameters**

- n\_groups (int) Number of groups in the population.
- **pool\_size** (*float*) Upperbound on the range of the existing population groups. A random value from 0 to the upperbound value will be assigned to each group.

run (\*args)

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### Notes

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use multiprocessing:

```
try: pickle.dumps(self.state)
    except pickle.PicklingError: args[1].put((arg[0],None))
    else: args[1].put((arg[0],self.state))
at the bottom of the user defined run() function.
```

4.2. arrested module 18

Warning: This function must be overridden by all Cortix modules

### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 4.3 community module

Community Cortix module used to model criminal group population in a community system. Community here is the system at large with all possible adult individuals included in a society.

#### **Notes**

These are the *port* names available in this module to connect to respective modules: *probation*, *adjudication*, *jail*, *prison*, *arrested*, and *parole*. See instance attribute *port\_names\_expected*.

```
__init__ (n_groups=1, non_offender_adult_population=100, offender_pool_size=0.0, free_offender_pool_size=0.0)
```

### **Parameters**

- n\_groups (int) Number of groups in the population.
- non\_offender\_adult\_population (float) Pool of individuals reaching the adult age (SI) unit. Default: 100.
- **offender\_pool\_size** (float) Upperbound on the range of the existing population groups. A random value from 0 to the upperbound value will be assigned to each group. This is typically a small number, say a fraction of a percent.

run (\*args)

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### **Notes**

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

```
try: pickle.dumps(self.state)
except pickle.PicklingError: args[1].put((arg[0],None))
```

```
else: args[1].put((arg[0],self.state))
```

at the bottom of the user defined *run()* function.

Warning: This function must be overridden by all Cortix modules

#### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm, self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 4.4 jail module

```
class jail.Jail(n_groups=1, pool_size=0.0)
    Bases: cortix.src.module.Module
```

Jail Cortix module used to model criminal group population in a jail.

### Notes

These are the *port* names available in this module to connect to respective modules: *probation*, *adjudication*, *arrested*, *prison*, and *community*. See instance attribute *port* names expected.

```
___init___(n_groups=1, pool_size=0.0)
```

### **Parameters**

- **n\_groups** (*int*) Number of groups in the population.
- **pool\_size** (*float*) Upperbound on the range of the existing population groups. A random value from 0 to the upperbound value will be assigned to each group.

run (\*args)

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### **Notes**

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

```
try: pickle.dumps(self.state)
except pickle.PicklingError: args[1].put((arg[0],None))
else: args[1].put((arg[0],self.state))
```

4.4. jail module 20

at the bottom of the user defined *run()* function.

Warning: This function must be overridden by all Cortix modules

#### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 4.5 parole module

```
class parole.Parole(n_groups=1, pool_size=0.0)
    Bases: cortix.src.module.Module
```

Parole Cortix module used to model criminal group population in a parole system.

#### **Notes**

These are the *port* names available in this module to connect to respective modules: *prison* and *community*. See instance attribute *port\_names\_expected*.

```
run (*args)
```

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self), \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### **Notes**

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

at the bottom of the user defined *run()* function.

```
try: pickle.dumps(self.state)
except pickle.PicklingError: args[1].put((arg[0],None))
else: args[1].put((arg[0],self.state))
```

**Warning:** This function must be overridden by all Cortix modules

### **Parameters**

- arg[0] (int) Index of the state in the communication queue.
- arg[1] (multiprocessing.Queue) When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 4.6 prison module

```
class prison.Prison(n_groups=1, pool_size=0.0)
    Bases: cortix.src.module.Module
```

Prison Cortix module used to model criminal group population in a prison.

### **Notes**

These are the *port* names available in this module to connect to respective modules: *parole*, *adjudication*, *jail*, and *community*. See instance attribute *port\_names\_expected*.

```
___init___(n_groups=1, pool_size=0.0)
```

#### **Parameters**

- **n\_groups** (*int*) Number of groups in the population.
- **pool\_size** (float) Upperbound on the range of the existing population groups. A random value from 0 to the upperbound value will be assigned to each group.

run (\*args)

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### Notes

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use\_multiprocessing:

```
try: pickle.dumps(self.state)
except pickle.PicklingError: args[1].put((arg[0],None))
else: args[1].put((arg[0],self.state))
at the bottom of the user defined run() function.
```

Warning: This function must be overridden by all Cortix modules

### **Parameters**

• arg[0] (int) - Index of the state in the communication queue.

4.6. prison module

• arg[1] (multiprocessing.Queue) — When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 4.7 probation module

```
class probation.Probation(n_groups=1, pool_size=0.0)
Bases: cortix.src.module.Module
```

Probation Cortix module used to model criminal group population in a probation.

### **Notes**

These are the *port* names available in this module to connect to respective modules: *adjudication*, *jail*, *arrested*, and *community*. See instance attribute *port names expected*.

```
___init__(n_groups=1, pool_size=0.0)
```

### **Parameters**

- n\_groups (int) Number of groups in the population.
- **pool\_size** (float) Upperbound on the range of the existing population groups. A random value from 0 to the upperbound value will be assigned to each group.

run (\*args)

Module run function

Run method with an option to pass data back to the parent process when running in Python multiprocessing mode. If the user does not want to share data with the parent process, this function can be overriden with run(self) or run(self, \*args) as long as self.state = None. If self.state points to anything but None, the user must use 'run(self, \*args).

### Notes

When in multiprocessing, \*args has two elements: comm\_idx and comm\_state. To pass back the state of the module, the user should insert the provided index comm\_idx and the state into the queue as follows:

### if self.use multiprocessing:

```
try: pickle.dumps(self.state)
    except pickle.PicklingError: args[1].put((arg[0],None))
    else: args[1].put((arg[0],self.state))
at the bottom of the user defined run() function.
```

Warning: This function must be overridden by all Cortix modules

### **Parameters**

• arg[0] (int) – Index of the state in the communication queue.

• arg[1] (multiprocessing.Queue) — When using the Python multiprocessing library state\_comm must have the module's self.state in it. That is, state\_comm.put((idx\_comm,self.state)) must be the last command in the method before return. In addition, self.state must be pickle-able.

# 4.8 run\_city\_justice module

Crimninal justice network dynamics modeling.

### This example uses 7 modules:

- Community
- Arrested
- Adjudication
- Jail
- Prison
- Probation
- Parole

and a population balance model is used to follow the offenders population groups between modules.

To run this case using MPI you should compute the number of processes as follows:

```
nprocs = 7 + 1 cortix
```

then issue the MPI run command as follows (replace *nprocs* with a number):

```
mpiexec -n nprocs run_justice.py
```

To run this case with the Python multiprocessing library, just run this file at the command line as

```
run_city_justice.py
run_city_justice.main()
    Cortix run file for a criminal justice network.

run_city_justice.n_groups
    int - Number of population groups being followed. This must be the same for all modules.

run_city_justice.end_time
    float - End of the flow time in SI unit.

run_city_justice.time_step
    float - Size of the time step between port communications in SI unit.

run_city_justice.use_mpi
    bool - If set to True use MPI otherwise use Python multiprocessing.
```

**CHAPTER** 

**FIVE** 

# **SUPPORT**

# 5.1 periodictable module

Properties of the chemical elements.

Each chemical element is represented as an object instance. Physicochemical and descriptive properties of the elements are stored as instance attributes.

```
Author Christoph Gohlke Version 2015.01.29
```

Radiochemical data (isotopes) has been added to this table (2015-2016) Origin: http://www.radiochemistry.org/ Valmor F. de Almeida: dealmeidavf@gmail.com; dealmeidav@ornl.gov

# 5.1.1 Requirements

• CPython 2.7 or 3.4

### References

- 1. http://physics.nist.gov/PhysRefData/Compositions/
- 2. http://physics.nist.gov/PhysRefData/IonEnergy/tblNew.html
- 3. http://en.wikipedia.org/wiki/%(element.name)s
- 4. http://www.miranda.org/~jkominek/elements/elements.db

### **Examples**

```
>>> from elements import ELEMENTS
>>> len(ELEMENTS)
109
>>> str(ELEMENTS[109])
'Meitnerium'
>>> ele = ELEMENTS['C']
>>> ele.number, ele.symbol, ele.name, ele.eleconfig
(6, 'C', 'Carbon', '[He] 2s2 2p2')
>>> ele.eleconfig_dict
{(1, 's'): 2, (2, 'p'): 2, (2, 's'): 2}
```

(continues on next page)

(continued from previous page)

```
>>> sum(ele.mass for ele in ELEMENTS)
14659.1115599
>>> for ele in ELEMENTS:
... ele.validate()
... ele = eval(repr(ele))
```

# 5.2 phase module

Phase *history* container. When you think of a phase value, think of that value at a specific point in time. This container holds the historic data of a phase; its species and quantities. This implementation treats access of time stamps within a tolerance. All searches for time stamped values are subjected to an approximation of the time stamp to avoid storing values too close to each other in time, and/or to return the closest value in time searched or no value if none can be found according to the tolerance.

# 5.2.1 Background

TODO: ATTENTION: The species (list of Specie) AND quantities (list of Quantity) data members have ARBITRARY density values either at an arbitrary point in the history or at no point in the history. This needs to be removed in the future to avoid confusion.

To obtain history values, associated to the phase, at a particular point in time, use the GetValue() method to access the history data frame (pandas) via columns and rows. ALERT: The corresponding values in species and quantities are OVERRIDEN and NOT to be used through the phase interface.

Author: Valmor F. de Almeida dealmeidav@ornl.gov; vfda Sat Sep 5 01:26:53 EDT 2015

Cortix: a program for system-level modules coupling, execution, and analysis.

```
class phase.Phase(time_stamp=None, time_unit=None, species=None, quantities=None)

Bases: object
```

Phase *history* container. A *Phase* consists of *Species* and *Quantities* varying with time. This container is meant to reproduce the basic idea of a material phase.

### AddQuantity (newQuant)

Adds a new quantity object to the dataframe. See quantity py for more details on the quantity class.

```
Parameters newQuant (object) -
```

```
AddRow (try_time_stamp, row_values)
```

Adds a row to the dataframe, with a timestamp of try\_time\_stamp and row values equal to row\_values. Take care that the dimensions and order of the data matches up!

### **Parameters**

```
try_time_stamp(float)-row_values(list)-
```

### AddSpecie (new\_specie)

Adds a new specie object to the phase history. See species.py for more details on the specie class.

```
Parameters new_specie (obj)-
```

```
ClearHistory (value=0.0)
```

Set species and quantities of history to a given value (default to zero value), all time stamps are preserved.

```
Parameters value (float) -
```

#### GetActors()

Returns a list of all the actors in the phase history.

Returns list(self.\_\_phase.colums)

Return type list

### GetColumn (actor)

Returns an entire column of data. A column is the entire history of data associated with a specific actor.

```
Parameters actor (str) -
```

Returns list(self.\_\_phase.loc[

Return type , actor]): list

### GetQuantities()

Returns the list of *Quantities*. The values in each *Quantity* are synchronized with the *Phase* data frame.

### Returns quantities

Return type list

### GetQuantity(name)

Returns the quantity evaluated at the last time step of the phase history. This also updates the value of the quantity object. If the quantity name does not exist the return is None.

```
Parameters name (str) -
```

### GetRow (try\_time\_stamp=None)

Returns an entire row of the phase dataframe. A row is a series of values that are all at the same time stamp.

```
Parameters try_time_stamp(float)-
```

Returns list(self.\_\_phase.loc[time\_stamp,

**Return type** ]): list

### GetSpecie (name)

Returns the species specified by name if it exists, or none if it doesn't.

```
Parameters name (str) -
```

Returns specie

Return type str

# GetSpecies()

Returns every single species in the phase history.

**Returns species** 

Return type list

### GetTimeStamps()

Returns a list of all the time stamps in the phase history.

Returns timeStamps

Return type list

### GetValue (actor, try\_time\_stamp=None)

Deprecated: use get\_value()

### ResetHistory(try time stamp=None, value=None)

Set species and quantities of history to a given value (default to zero value) only one time stamp is preserved (default to last time stamp).

### **Parameters**

- try\_time\_stamp(float)-
- value (float) -

### ScaleRow (try\_time\_stamp, value)

Multiplies all of the data in a row (except time stamp) by a scalar value.

### **Parameters**

- try\_time\_stamp (float) -
- value (float) -

### SetSpecieId (name, val)

Sets the flag of a specie "name" equal to val.

### **Parameters**

- name (str) -
- **val** (int) -

### SetValue (actor, value, try\_time\_stamp=None)

For the record: old def SetValue(self, time\_stamp, actor, value):

#### **Parameters**

- actor (str) -
- value (float) -
- try\_time\_stamp (float) -

### WriteHTML (fileName)

Convert the *Phase* container into an HTML file.

```
Parameters fileName (str) -
```

```
__init__ (time_stamp=None, time_unit=None, species=None, quantities=None)
```

Sometimes an empty Phase object is created by user code. This case needs adequate logic for None types. Note on usage: when passing quantities, do set the value argument explicitly to help define the type and avoid SetValue() errors with Pandas. This is to be investigated later. Also, the usage of a DataFrame needs to be re-evaluated. Maybe better to use a Quantity object and a Specie object with a Pandas Series history as a value to avoid the existance of a value in Quantity and a value in Phase that are not in sync.

# get\_quantity (name, try\_time\_stamp=None)

New version. Get the quantity *name* at a point in time closest to *try\_time\_stamp* up to a tolerance. If no time stamp is passed, the whole history is returned.

### **Parameters**

- name (str)-
- **try\_time\_stamp** (*float*, *int* or *None*) Time stamp of desired quantity value. Default: None returns the whole quantity history.

### Returns quant.value

Return type float or int or other

### get\_quantity\_history(name)

Create a Quantity *name* history. This will create a fully qualified Quantity object and return to the caller. The function is typically needed for data output to a file through *pickle*. Since the value attribute of a quantity can be any data structure, a time-series is built on the fly and stored in the value attribute. In addition the time unit is added to the final return value as a tuple.

```
Parameters name (str) –

Returns quant_history

Return type tuple(Quantity,str)
```

# get\_value (actor, try\_time\_stamp=None)

Returns the value associated with a specified actor at a specified time stamp.

### **Parameters**

```
• actor (str)-
```

• try\_time\_stamp(float)-

Returns self.\_\_phase.loc[time\_stamp, actor]

Return type float

### has\_time\_stamp (try\_time\_stamp)

Checks to see if try\_time\_stamp exists in the phase history.

Parameters try\_time\_stamp -

### quantities

Returns the list of *Quantities*. The values in each *Quantity* are synchronized with the *Phase* data frame.

### Returns quantities

**Return type** list

### set\_value (actor, value, try\_time\_stamp=None)

New version. Discontinue using SetValue()

### species

Returns every single species in the phase history.

Returns species

**Return type** list

# $\verb|timeStamps||$

Returns a list of all the time stamps in the phase history.

Returns timeStamps

**Return type** list

### time\_stamps

Get all time stamps in the index of the data frame.

Returns time\_stamps

**Return type** list

### time\_unit

Returns the time unit of the Phase.

Returns time\_unit

Return type str

# 5.3 quantity module

```
Author: Valmor de Almeida dealmeidav@ornl.gov; vfda
This Quantity class is to be used with other classes in plant-level process modules.
For unit testing do at the linux command prompt: python quantity.py
Sat Sep 5 12:51:34 EDT 2015
class quantity.Quantity(name='null-quantity',
                                                            formalName='null-quantity',
                                                                                              value=0.0,
                                 unit='null-unit')
     Bases: object
     todo: this probably should not have a "value" for the same reason as Specie. this needs some thinking.
     well not so fast. This can be used to build a quantity with anything as a value. For instance a history of the
     quantity as a time series.
     GetFormalName()
           Returns the formal name of the quantity.
               Returns formalName
               Return type str
     GetUnit()
           Returns the units of the quantity.
               Returns unit
               Return type str
     GetValue()
          Gets the numerical value of the quantity.
               Returns value
               Return type any type
     SetFormalName(fn)
           Sets the formal name of the property to fn.
               Parameters fn (str) -
     SetName (n)
           Sets the name of the quantity in question to n.
               Parameters n (str) -
     SetUnit(f)
           Sets the units of the quantity to f (for example, density would be in units of g/cc.
               Parameters f (str) -
     SetValue (v)
           Sets the numerical value of the quantity to v.
               Parameters v (float) -
      repr ()
           Used to print the data stored by the quantity class. Will print out name, formal name, the value of the
           quantity and its unit.
               Returns s
```

Return type str

```
__str__()
```

Used to print the data stored by the quantity class. Will print out name, formal name, the value of the quantity and its unit.

Returns s

Return type str

#### formalName

Returns the formal name of the quantity.

Returns formalName

Return type str

#### formal name

Returns the formal name of the quantity.

Returns formalName

Return type str

#### get name()

Returns the name of the quantity.

Returns name

Return type str

#### name

Returns the name of the quantity.

Returns name

Return type str

This will support a few possibities for data storage in the self.\_\_value member.

Pandas Series. If self.\_\_value is a Pandas Series, plot against the index. However the type stored in the Series matter. Suppose it is a series of a *numpy* array. This must be of the same rank for every entry. This plot method assumes it is an iterable type of the same length for every entry in the series. A plot of all elements in the type against the index of the series will be made. The plot may have all elements in one axis or each element in its own axis.

### unit

Returns the units of the quantity.

Returns unit

Return type str

### value

Gets the numerical value of the quantity.

Returns value

Return type any type

# 5.4 specie module

Author: Valmor de Almeida dealmeidav@ornl.gov; vfda

5.4. specie module 31

This Specie class is to be used with other classes in plant-level process modules.

NB: Species is always used either in singular or plural cases, the class named here reflects one species. If many species are used in an external context, the species object name can be used without conflict.

For unit testing do at the linux command prompt: python specie.py

NB: The Specie() class encapsulates either the molecular or empirical chemical formula of a compound. This is done as follows. Say MAO2 is either a molecular or empirical chemical formula of a ficticious compound denoting minor actinides dioxide. The list of atoms is given as follows:

```
['0.49*Np-237', '0.42*Am-241', '0.08*Am-243', '0.01*Cm-244', '2.0*O-16']
```

note the MA forming nuclides add to 1 = 0.49 + 0.42 + 0.08 + 0.01. Therefore the number of atoms in this compound is 3. 1 MA "atom" and 2 O. Note that the total number of "atoms" is obtained by summing all multipliers: 0.49 + 0.42 + 0.08 + 0.01 + 2.0. The nuclide is indicated by the element symbol followed by a dash and the atomic mass number. Here the number of nuclide types is 5 (self.\_nNuclideTypes).

The numbers preceding the nuclide symbol before the \* will be referred to as multipliers. The sum of the multipliers will add to the number of "atoms" in the formula. WARNING: a multiplier could be in the format 0.00e-00. In this case a hiphen may appear twice, e.g.: 1.549e-09\*U-233

Other forms can be used for common true species

```
['Np-237', '2.0*O-16'] or ['Np-237', 'O-16', 'O-16'] or [ '2*H', 'O' ] or [ 'H', 'O', 'H' ] etc...
```

This code will calculate the molar mass of any species with a given valid atom list using a provided periodic table of chemical elements. The user can also reset the value of the molar mass with a setter method.

Sat May 9 21:40:48 EDT 2015 created; vfda

```
class specie. Specie (name='null', formula_name='null', phase='null', atoms=[], molarCC=0.0,
                         massCC=0.0, flag=None)
     Bases: object
```

todo: phase should not be here; concentrations should not be here only molar quantities should be here see the Phase container

```
GetAtoms()
```

### GetFlag()

Returns the flag associated with the species.

### Returns flag

Return type str

### GetFormula()

Returns the molecular or empirical formula of the species. It is usually a list, for example, of the form ['2\*H', 'O'].

Returns formula

Return type list

### GetFormulaName()

Returns the formulaic name of the compound. For example, "Dihydrogen monoxide".

Returns self. formula name

Return type str

### GetMassCC()

Returns the numerical value of the mass density of the species (mass/volume).

Returns massCC

### Return type float

### GetMassCCUnit()

Returns the unit used to measure the mass density of the species.

### Returns massCCUnit

Return type str

### GetMolarCC()

Returns the numerical value for the number (molar) density of the species (moles/volume).

### Returns molarCC

Return type float

### GetMolarCCUnit()

Returns the unit used to measure molar density of the species.

### Returns molarCCUnit

Return type str

### GetMolarGammaPwr()

Returns the amount of gamma radiation produced per mole of this species (measured in units of power).

### Returns molarGammaPwr

Return type float

### GetMolarGammaPwrUnit()

Returns the unit used to measure the amount of gamma radiation produced per mole of this species.

### Returns molarGammaPwrUnit

Return type str

### GetMolarHeatPwr()

Returns the amount of heat generated per mole of this species.

### Returns molarHeatPwr

Return type float

### GetMolarHeatPwrUnit()

Returns the unit used to measure the amount of heat generated per mole of this species.

### Returns molarHeatPwrUnit

Return type str

### GetMolarMass()

Returns the numerical value for the molar mass of the species. Units are given by molarMassUnit.

### Returns molarMass

Return type float

### GetMolarMassUnit()

Returns the unit used to measure the molar mass of the species.

### Returns molarMassUnit

Return type str

### GetMolarRadioactivity()

Returns the numerical value for molar radioactivity of the species.

5.4. specie module 33

## Returns molarRadioactivity

Return type float

# GetMolarRadioactivityFractions()

Returns a list of numbers that speciefies the % of molar reactivity that comes from each type of atom in the species. For example, a molarRadioactivityFraction of [0.65, 0.35] for water means that 65% of the molar radioactivity comes from the hydrogen atoms and 35% comes from the oxygen atom.

## Returns molarRadioactivityFractions

**Return type** list

#### GetMolarRadioactivityUnit()

Returns the unit used to measure molar radioactivity.

#### Returns molarRadioactivityUnit

Return type str

## GetNAtoms()

Returns the total number of atoms comprising the species. For example, water is comprised of three atoms.

#### Returns nAtoms

Return type int

## GetNNuclideTypes()

Returns the number of different types of atoms comprising the species. For example, water is composed of two different types of atoms, hydrogen and oxygen.

#### Returns nNuclideTypes

Return type int

## GetName()

Returns the empirical name of the species. For example, "water".

#### Returns name

Return type str

#### GetPhase()

Returns the phase history of the species.

## Returns phase

Return type dataFrame

SetAtoms (atoms)

#### SetFlag(f)

Sets the flag associated with the species to f.

```
Parameters f (str)-
```

## SetFormula (atoms)

Sets the species' formula equal to atoms. Will automatically update the molar mass of the species, and will also fail if atoms is not a list of strings.

Parameters atoms (list) -

## ${\tt SetFormulaName}\,(f)$

Sets the formulaic name to f.

## Returns self. formula name

5.4. specie module 34

#### Return type str

#### SetMassCC(v)

Sets the numerical value of the mass density equal to v.

Parameters v(float)-

#### SetMassCCUnit(v)

Sets the units used to measure mass density to v.

Parameters v(str)-

#### SetMolarCC(v)

Sets the numerical value for the molar density of the species to v.

Parameters v(float) -

#### SetMolarCCUnit(v)

Sets the unit used to measure the molar density of the species to v.

Parameters v(str) -

#### SetMolarGammaPwr(v)

Sets the amount of gamma radiation produced per mole of this species to v.

Parameters v(float)-

#### SetMolarGammaPwrUnit (v)

Sets the unit used to measure the amount of gamma radiation produced per mole of this species to v.

Parameters v(str)-

#### SetMolarHeatPwr(v)

Sets the amount of heat generated per mole of this species to v.

**Parameters** v (float) -

#### SetMolarHeatPwrUnit(v)

Sets the unit used to measure the amount of heat generated per mole of this species to v.

Parameters v(str)-

#### SetMolarMass(v)

Sets the molar mass of the species equal to v.

Parameters v(float)-

#### SetMolarMassUnit(v)

Sets the unit used to measure the molar mass of the species to v.

Parameters v(str)-

#### SetMolarRadioactivity(v)

Sets the molar radioactivity of the species equal to v.

Parameters v(float)-

### SetMolarRadioactivityFractions (fracs)

Sets molarRadioactivityFractions equal to fracs. Fracs must be a list of floatswith the same length as there are different atoms in the species, or the function call will fail. (e.g. self.\_atoms and fracs must be of the same length). Take care to ensure that the elements of fracs match with the elements of self.\_atoms! (65% is in the same position in fracs as hydrogen is in self.\_atoms, following the above example).

Parameters fracs (list) -

5.4. specie module

#### SetMolarRadioactivityUnit(v)

Sets the unit used to measure molar radioactivity to v.

Parameters v(str) –

#### SetName(n)

Sets the empirical name of the species to n.

Parameters n (str) -

#### SetPhase(p)

Sets the phase history to p.

Parameters p (dataFrame) -

#### atoms

#### flag

Returns the flag associated with the species.

Returns flag

Return type str

#### formula

Returns the molecular or empirical formula of the species. It is usually a list, for example, of the form ['2\*H', 'O'].

Returns formula

**Return type** list

## formula\_name

Returns the formulaic name of the compound. For example, "Dihydrogen monoxide".

Returns self.\_\_formula\_name

Return type str

#### massCC

Returns the numerical value of the mass density of the species (mass/volume).

Returns massCC

Return type float

#### massCCUnit

Returns the unit used to measure the mass density of the species.

Returns massCCUnit

Return type str

## molarCC

Returns the numerical value for the number (molar) density of the species (moles/volume).

Returns molarCC

Return type float

#### molarCCUnit

Returns the unit used to measure molar density of the species.

Returns molarCCUnit

Return type str

#### molarGammaPwr

Returns the amount of gamma radiation produced per mole of this species (measured in units of power).

### Returns molarGammaPwr

Return type float

## molarGammaPwrUnit

Returns the unit used to measure the amount of gamma radiation produced per mole of this species.

#### Returns molarGammaPwrUnit

Return type str

#### molarHeatPwr

Returns the amount of heat generated per mole of this species.

#### Returns molarHeatPwr

Return type float

#### molarHeatPwrUnit

Returns the unit used to measure the amount of heat generated per mole of this species.

#### Returns molarHeatPwrUnit

Return type str

#### molarMass

Returns the numerical value for the molar mass of the species. Units are given by molarMassUnit.

#### Returns molarMass

Return type float

## molarMassUnit

Returns the unit used to measure the molar mass of the species.

#### Returns molarMassUnit

Return type str

#### molarRadioactivity

Returns the numerical value for molar radioactivity of the species.

## Returns molarRadioactivity

Return type float

## molarRadioactivityFractions

Returns a list of numbers that speciefies the % of molar reactivity that comes from each type of atom in the species. For example, a molarRadioactivityFraction of [0.65, 0.35] for water means that 65% of the molar radioactivity comes from the hydrogen atoms and 35% comes from the oxygen atom.

## Returns molarRadioactivityFractions

Return type list

#### molarRadioactivityUnit

Returns the unit used to measure molar radioactivity.

## Returns molarRadioactivityUnit

Return type str

## nAtoms

Returns the total number of atoms comprising the species. For example, water is comprised of three atoms.

5.4. specie module 37

```
Returns nAtoms
```

Return type int

## nNuclideTypes

Returns the number of different types of atoms comprising the species. For example, water is composed of two different types of atoms, hydrogen and oxygen.

```
Returns nNuclideTypes
```

Return type int

#### name

Returns the empirical name of the species. For example, "water".

Returns name

Return type str

## phase

Returns the phase history of the species.

Returns phase

**Return type** dataFrame

# 5.5 stream module

Author: Valmor F. de Almeida dealmeidav@ornl.gov; vfda

Stream container

VFdALib support classes

Sat Aug 15 17:24:02 EDT 2015

**class** stream.**Stream**(timeStamp, species=None, quantities=None, values=0.0)

Bases: object

## GetActors()

Returns the actors present in the stream of data.

Returns list(self.stream.columns)

Return type list

### GetQuantities()

Returns all the quantities given by the stream.

Returns self.quantities

Return type list

#### GetQuantity (name)

Returns the specified quantity called "name" from the stream, or none if the specified name does not exist.

Parameters name (str) -

Returns quant

Return type float

5.5. stream module 38

#### GetRow (timeStamp=None)

Returns an entire row of data from the stream. A row of data is all the data in a dataframe at a specified time stamp, given by timeStamp. If timeStamp is not specified, this function will return the entire stream dataframe.

```
Parameters timeStamp(float)-
```

#### Returns

- self.stream.loc[self.timestamp, (]) or self.stream.loc[timeStamp, :]):)
- list

## GetSpecie (name)

Returns a specie named "name" from the stream.

```
Parameters name (str) -
```

Returns specie

Return type obj

#### GetSpecies()

Returns a list of all species in the stream.

## Returns self.species

Return type list

## GetTimeStamp()

Returns the time stamp of the stream.

## Returns self.timeStamp

Return type float

## GetValue (actor, timeStamp=None)

Returns the value associated with a specified "actor" at a specified "timeStamp". If no timeStamp is specified, then the function will return all values associated with the specified actor at all time stamps.

#### **Parameters**

- actor (str) -
- timeStamp (float) -

#### **Returns**

- self.stream.loc[self.timeStamp, actor] or self.stream.loc[timeStamp,
- actor] (list or float, respectively.)

#### SetSpecieId (name, val)

Sets the numerical id of the specie of name "name" to val.

### **Parameters**

- name (str)-
- **val** (int) -

## **SetValue** (actor, value=None, timeStamp=None)

Sets the value associated with a specified actor at a specified timeStamp to "value". If no value is specified, the value will default to 0.0. If no timeStamp is specified, it will set all values associated with actor to the specified value (or 0.0 if value = None).

#### **Parameters**

5.5. stream module 39

- actor (str)-
- value(float)-
- timeStamp(float)-

5.5. stream module 40

**CHAPTER** 

SIX

# **NUCLEAR**

# 6.1 actor module

This is a simple way to hide the name of species of interest in a simulation. The user would modify and copy this class into the Cortix module of interest and keep it private. Author: Valmor de Almeida dealmeidav@ornl.gov; vfda Sat Aug 15 13:41:12 EDT 2015

```
class actor.Actor(name)
Bases: object
See atoms list in Specie.

atoms
Returns the specific nuclides found in the specified chemical.

Returns atoms
Return type list(str)

formula
Returns the formula of the chemical in question.

Returns formula
Return type str
```

# 6.2 fuel bucket module

Author: Valmor de Almeida dealmeidav@ornl.gov; vfda

```
class fuel_bucket.FuelBucket (specs=Empty DataFrame Columns: [] Index: [])
    Bases: object
    __repr__()
        Converts to string.
    __str__()
        Converts to string.
```

## cladding\_end\_thickness

Gets the thickness of the hemispherical cladding end caps that are placed on the top and bottom of the fuel slug, in cm.

## Returns cladding\_end\_thickness

Return type float

## cladding\_mass

Returns the total mass of cladding material in the bucket, in grams.

Returns cladding\_mass

Return type float

## cladding\_phase

Returns the phase history of the cladding.

Returns cladding\_phase

**Return type** dataFrame

## cladding\_volume

Returns the total volume of cladding in the bucket, in cm<sup>3</sup>.

Returns cladding\_volume

Return type float

## cladding\_wall\_thickness

Returns the thickness of the cladding wall which is on the outside of every fuel slug, and in between both sections of fuel, in cm.

Returns cladding\_wall\_thickness

Return type float

## fresh\_u235\_mass

Returns the total amount of uranium-235 in the bucket, in grams.

Returns fresh\_u235\_mass

Return type float

## fresh\_u238\_mass

Returns the total amount of uranium-238 in the bucket, in grams.

Returns fresh\_u238\_mass

**Return type** float

# ${\tt fresh\_u\_mass}$

Returns the total amount of uranium in the bucket, in grams.

Returns fresh u mass

Return type float

## fuel enrichment

Returns the enrichment of the fuel slugs in the bucket, in %.

Returns fuel\_enrichment

Return type float

#### fuel\_mass

Returns the total mass of fuel in the solid phase in the bucket.

Returns fuel\_mass

Return type float

#### fuel mass unit

Returns the unit that is used to measure the mass of fuel in the bucket.

Returns fuel\_mass\_unit

Return type str

#### fuel\_phase

Returns the phase history of the fuel.

Returns fuel\_phase

**Return type** pandas.core.frame.DataFrame

#### fuel\_radioactivity

Returns the total radioactivity of the solid phase fuel, in units of curies.

Returns fuel\_radioactivity

Return type float

## fuel\_volume

Returns the total volume of fuel in the entire bucket, in cm<sup>3</sup>.

Returns fuel volume

Return type float

#### gamma\_pwr

Returns the amount of gamma radiation given off by the fuel bucket, in units of watts.

Returns gamma pwr

Return type float

## get\_cladding\_end\_thickness()

Gets the thickness of the hemispherical cladding end caps that are placed on the top and bottom of the fuel slug, in cm.

Returns cladding\_end\_thickness

Return type float

#### get\_cladding\_mass()

Returns the total mass of cladding material in the bucket, in grams.

Returns cladding\_mass

Return type float

#### get cladding phase()

Returns the phase history of the cladding.

Returns cladding\_phase

Return type dataFrame

## get\_cladding\_volume()

Returns the total volume of cladding in the bucket, in cm<sup>3</sup>.

Returns cladding\_volume

Return type float

## get\_cladding\_wall\_thickness()

Returns the thickness of the cladding wall which is on the outside of every fuel slug, and in between both sections of fuel, in cm.

```
Returns cladding_wall_thickness
         Return type float
get_fresh_u235_mass()
     Returns the total amount of uranium-235 in the bucket, in grams.
         Returns fresh u235 mass
         Return type float
get_fresh_u238_mass()
     Returns the total amount of uranium-238 in the bucket, in grams.
         Returns fresh_u238_mass
         Return type float
get_fresh_u_mass()
     Returns the total amount of uranium in the bucket, in grams.
         Returns fresh_u_mass
         Return type float
get_fuel_enrichment()
     Returns the enrichment of the fuel slugs in the bucket, in %.
         Returns fuel_enrichment
         Return type float
get_fuel_mass()
     Returns the total mass of fuel in the solid phase in the bucket.
         Returns fuel_mass
         Return type float
get_fuel_mass_unit()
     Returns the unit that is used to measure the mass of fuel in the bucket.
         Returns fuel_mass_unit
         Return type str
get fuel phase()
     Returns the phase history of the fuel.
         Returns fuel_phase
         Return type pandas.core.frame.DataFrame
get fuel radioactivity()
     Returns the total radioactivity of the solid phase fuel, in units of curies.
         Returns fuel_radioactivity
         Return type float
get_fuel_volume()
     Returns the total volume of fuel in the entire bucket, in cm<sup>3</sup>.
```

Returns fuel\_volume
Return type float

```
get_gamma_pwr()
     Returns the amount of gamma radiation given off by the fuel bucket, in units of watts.
         Returns gamma_pwr
         Return type float
get heat pwr()
     Returns the total amount of heat generated by the bucket, in units of watts.
         Returns heat_pwr
         Return type float
get_inner_slug_id()
     Returns the inner diameter of the inner section of fuel, in cm.
         Returns inner slug id
         Return type float
get_inner_slug_od()
     Returns the outer diameter of the inner section of fuel, in cm.
         Return type float
```

```
Returns inner slug od
```

get\_n\_slugs()

Returns the number of fuel slugs in the bucket.

```
Returns n slugs
        Return type int
get_name()
```

Returns the name of the fuel bucket.

```
Returns name
        Return type str
get_outer_slug_id()
```

Returns the inner diameter of the outer section of fuel, in cm.

```
Returns outer_slug_id
        Return type float
get_outer_slug_od()
```

Returns the outer diameter of the outer section of fuel, in cm. A fuel slug consists of an outer section of fuel and an inner section of fuel, with cladding on the outside of the slug and between the inner and outer sections of fuel.

```
Returns outer_slug_od
        Return type float
get_radioactivity()
```

Returns the radioactivity of the fuel bucket, in units of curies.

```
Returns radioactivity
         Return type float
get slug cladding volume()
     Returns the volume of cladding present in a single fuel slug, in cm<sup>3</sup>.
```

```
Returns slug_cladding_volume
         Return type float
get_slug_fuel_volume()
     Returns the volume of fuel present in a single fuel slug, in cm<sup>3</sup>.
         Returns slug fuel volume
         Return type float
get_slug_length()
     Returns the length of each slug in the fuel bucket.
         Returns slug_length
         Return type float
get_slug_type()
     Returns the type of slugs being stored in the bucket (inner slug or outer slug).
         Returns slug_type
         Return type str
heat_pwr
     Returns the total amount of heat generated by the bucket, in units of watts.
         Returns heat_pwr
         Return type float
inner_slug_id
     Returns the inner diameter of the inner section of fuel, in cm.
         Returns inner_slug_id
         Return type float
inner_slug_od
     Returns the outer diameter of the inner section of fuel, in cm.
         Returns inner_slug_od
         Return type float
n slugs
     Returns the number of fuel slugs in the bucket.
         Returns n_slugs
         Return type int
name
     Returns the name of the fuel bucket.
         Returns name
         Return type str
outer_slug_id
     Returns the inner diameter of the outer section of fuel, in cm.
         Returns outer_slug_id
         Return type float
```

## outer\_slug\_od

Returns the outer diameter of the outer section of fuel, in cm. A fuel slug consists of an outer section of fuel and an inner section of fuel, with cladding on the outside of the slug and between the inner and outer sections of fuel.

Returns outer\_slug\_od

Return type float

#### radioactivity

Returns the radioactivity of the fuel bucket, in units of curies.

Returns radioactivity

Return type float

#### set\_cladding\_phase(phase)

Set's the phase history to specific values.

Parameters phase (dataFrame) -

## set\_fuel\_phase(phase)

Sets the current fuel phase to a specified phase value.

Parameters phase (dataFrame) -

#### set slug length(x)

Sets the length of all slugs in the bucket to x. Used for chopping.

Parameters x (float) -

## slug\_cladding\_volume

Returns the volume of cladding present in a single fuel slug, in cm<sup>3</sup>.

Returns slug\_cladding\_volume

Return type float

## slug\_fuel\_volume

Returns the volume of fuel present in a single fuel slug, in cm<sup>3</sup>.

Returns slug\_fuel\_volume

Return type float

## slug\_length

Returns the length of each slug in the fuel bucket.

Returns slug\_length

Return type float

#### slug\_type

Returns the type of slugs being stored in the bucket (inner slug or outer slug).

Returns slug\_type

Return type str

# 6.3 fuel bundle module

This FuelBundle class is a container for usage with other plant-level process modules. It is meant to represent a fuel bundle of an oxide fuel LWR reactor. There are three main data structures:

- 1. fuel bundle specs
- 2. solid phase
- 3. gas phase

The container user will have to provide all the data and from then on, this class will help acess the data. The printing methods reveal the contained data.

Author: Valmor de Almeida dealmeidav@ornl.gov; vfda Sun Dec 27 15:06:55 EST 2015

## fresh\_u235\_mass

Returns the amount of uranium-235 in the bucket, in grams.

Returns fresh u235 mass

Return type float

#### fresh\_u238\_mass

Returns the amount of uranium-238 in the bucket, in grams.

Returns fresh\_u238\_mass

Return type float

#### fresh\_u\_mass

Returns the amount of uranium in the bundle, in grams.

Returns fresh u mass

Return type float

## fuel\_enrichment

Returns the enrichment of the fuel pins in the bundle, in %.

Returns fuel\_enrichment

Return type float

## fuel\_mass

Returns the total numerical value for mass of fuel in the solid phase in the bundle.

Returns fuel\_mass

Return type float

#### fuel mass unit

Returns the unit used to measure the mass of fuel in the bundle.

Returns fuel\_mass\_unit

Return type str

## fuel\_pin\_length

Returns the length of each fuel pin in the fuel bundle. A fuel pin is a cylindircal section of uranium fuel that is surrounded by cladding.

Returns fuel\_pin\_length

Return type float

## fuel\_pin\_radius

Returns the radius of the fuel pin, in cm.

#### fuel\_pin\_volume

Returns the volume of fuel in each fuel pin, in cm<sup>3</sup>.

## Returns fuel\_pin\_volume

Return type float

#### fuel radioactivity

Returns the total radioactivity of the fuel in the solid phase in the fuel bundle.

## Returns fuel\_radioactivity

Return type float

#### fuel rod od

Returns the outer diameter of the fuel rod, in cm. A fuel rod consists of a fuel pin surrounded by cladding.

#### Returns fuel rod od

Return type float

#### fuel\_volume

Returns the total volume of fuel in the bundle, in cm<sup>3</sup>.

#### Returns fuel volume

Return type float

#### gamma\_pwr

Returns the total amount of gamma radiation given by the fuel bundle, in watts.

## Returns gamma\_pwr

Return type float

## gas\_mass

Returns the total numerical value for mass of the fuel in the gas phase.

#### gas\_phase

Returns the gas phase history of the fuel.

## Returns gas\_phase

Return type dataFrame

## gas\_radioactivity

Returns the total radioactivity of the fuel in the gas phase in the fuel bundle, in curies.

# Returns gas\_radioactivity

Return type float

#### get\_fresh\_U235\_mass()

Returns the amount of uranium-235 in the bucket, in grams.

# Returns fresh\_u235\_mass

Return type float

#### get\_fresh\_u238\_mass()

Returns the amount of uranium-238 in the bucket, in grams.

## Returns fresh\_u238\_mass

Return type float

## get\_fresh\_u\_mass()

Returns the amount of uranium in the bundle, in grams.

#### Returns fresh\_u\_mass

Return type float

## get\_fuel\_enrichment()

Returns the enrichment of the fuel pins in the bundle, in %.

#### Returns fuel enrichment

Return type float

## get\_fuel\_mass()

Returns the total numerical value for mass of fuel in the solid phase in the bundle.

## Returns fuel\_mass

Return type float

## get\_fuel\_mass\_unit()

Returns the unit used to measure the mass of fuel in the bundle.

### Returns fuel\_mass\_unit

Return type str

## get\_fuel\_pin\_length()

Returns the length of each fuel pin in the fuel bundle. A fuel pin is a cylindircal section of uranium fuel that is surrounded by cladding.

## Returns fuel\_pin\_length

Return type float

## get\_fuel\_pin\_radius()

Returns the radius of the fuel pin, in cm.

## get\_fuel\_pin\_volume()

Returns the volume of fuel in each fuel pin, in cm<sup>3</sup>.

## Returns fuel\_pin\_volume

Return type float

#### get\_fuel\_radioactivity()

Returns the total radioactivity of the fuel in the solid phase in the fuel bundle.

## Returns fuel\_radioactivity

Return type float

#### get fuel rod od()

Returns the outer diameter of the fuel rod, in cm. A fuel rod consists of a fuel pin surrounded by cladding.

## Returns fuel\_rod\_od

Return type float

## ${\tt get\_fuel\_volume}\;(\;)$

Returns the total volume of fuel in the bundle, in cm<sup>3</sup>.

#### Returns fuel volume

Return type float

## get\_gamma\_pwr()

Returns the total amount of gamma radiation given by the fuel bundle, in watts.

#### Returns gamma pwr

```
Return type float
get_gas_mass()
     Returns the total numerical value for mass of the fuel in the gas phase.
get_gas_phase()
     Returns the gas phase history of the fuel.
         Returns gas_phase
         Return type dataFrame
get_gas_radioactivity()
     Returns the total radioactivity of the fuel in the gas phase in the fuel bundle, in curies.
         Returns gas_radioactivity
         Return type float
get_heat_pwr()
     Returns the total amount of heat produced by the fuel bundle, in watts.
         Returns heat pwr
         Return type float
get_n_fuel_rods()
     Returns the number of fuel rods in the bundle.
         Returns n fuel rods
         Return type int
get_name()
     Returns the name of the fuel bundle.
         Returns name
         Return type str
get_radioactivity()
     Returns the total radioactivity of the fuel bundle, in curies.
         Returns raduioactivity
         Return type float
get_solid_phase()
     Returns the solid phase history associated with this fuel bundle.
         Returns solidPhase
         Return type dataFrame
heat_pwr
     Returns the total amount of heat produced by the fuel bundle, in watts.
         Returns heat_pwr
         Return type float
n_fuel_rods
     Returns the number of fuel rods in the bundle.
         Returns n_fuel_rods
```

Return type int

#### name

Returns the name of the fuel bundle.

Returns name

Return type str

#### radioactivity

Returns the total radioactivity of the fuel bundle, in curies.

Returns raduioactivity

Return type float

#### set\_fuel\_pin\_length(x)

Sets the length of all fuel pins in the bundle to x.

Returns x

Return type float

## set\_gas\_phase(phase)

Sets the gas phase history of the fuel equal to phase.

Parameters phase (dataFrame) -

## set\_solid\_phase(phase)

Sets the solid phase history of the fuel equal to phase.

Parameters phase (dataFrame) -

#### solid phase

Returns the solid phase history associated with this fuel bundle.

Returns solidPhase

**Return type** dataFrame

# 6.4 fuel\_segment module

```
Fuel segment Author: Valmor de Almeida dealmeidav@ornl.gov; vfda Sat Jun 27 14:46:49 EDT 2015
```

```
class fuel_segment.FuelSegment (geometry=Series([], dtype: float64), species=[])
    Bases: object
```

```
__repr__()
```

Used to pront the geometry of the fuel segment and the species that it consists of.

Returns s

Return type str

```
__str__()
```

Used to print the geometry of the fuel segment and the species that it consists of.

Returns s

Return type str

## geometry

Returns the geometry of the fuel bundle (cylindrical, hexoganol, rectangular, etc).

Returns geometry

Return type str

#### get attribute(name, nuclide=None, series=None)

Used to get stored fuel segment properties, either overall (as an average), or on a nuclide basis. "name" in this case refers to the attribute in question. At this point in time, series is not implemented and passing it to this function will result in an error. Possible attributes that may be retrieved with this function, as well as the name to pass to this function to retrieve them are: number of segments in the bundle (n-segments, always equal to 1), the id of the segment that makes up the bundle (segment-id), the volume of the fuel in the bundle (fuel-volume), the total volume of the segment (segment-volume), the diameter (fuel-diameter) and length (fuel-length) of the segment, the mass or mass density of the segment (mass or mass-cc, respectively), or the total or per-volume radioactivity, gamma radiation density or heat density of the fuel segment (radioactivity and radioactivityDens, gamma and gamma-dens, and heat and heat-dens, respectively).

Finally, density or total mass of a specific nuclide can be determined by passing a specific nuclide to the function, with a name value of mass or mass-cc.

#### **Parameters**

- name (str) -
- nuclide (str) -

#### Returns

Return type many types

#### get\_geometry()

Returns the geometry of the fuel bundle (cylindrical, hexoganol, rectangular, etc).

## Returns geometry

Return type str

## get\_specie (name)

Returns a specie named [name] from the list of species making up the fuel bundle. If no name is specified, this function will return None.

```
Parameters name (str) -
```

Returns specie

Return type obj

## get\_species()

Returns the species object which describes the composition of the fuel bundle. The species encapsulates all chemical species present in the fuel bundle.

#### **Returns species**

Return type object

#### specie

Returns a specie named [name] from the list of species making up the fuel bundle. If no name is specified, this function will return None.

```
Parameters name (str) -
```

Returns specie

Return type obj

## species

Returns the species object which describes the composition of the fuel bundle. The species encapsulates all chemical species present in the fuel bundle.

#### Returns species

## Return type object

# 6.5 fuelsegmentsgroups module

Author: Valmor de Almeida dealmeidav@ornl.gov; vfda

Fuel segment

VFdALib support classes

Sat Jun 27 14:46:49 EDT 2015

```
\textbf{class} \hspace{0.1in} \texttt{fuelSegmentsGroups} \hspace{0.1in} (\textit{key=None}, \textit{fuelSegments=None}) \\
```

Bases: object

Creates a dictionary of lists of fuel segment objects, with the keys typically being timestamps. Each fuel segment object has two data members, a *Pandas* Series for geometry spec and a panda DataFrame for property density.

## AddGroup (key, fuelSegments=None)

Appends the dictionary with a new key and associated list of fuelSegments. If the specified key is already present in the dictionary, then the specified list of fuel segments will be appended to the list of fuel segments already associated with the specified key.

#### **Parameters**

- key (str) -
- fuelSegments (list) -

**GetAttribute** (groupKey=None, attributeName=None, nuclideSymbol=None, nuclideSeries=None)

Returns the average value of an attribute amongst all elements in a group (WARNING: keys with no values associated with them will lower this average!). If groupKey is not specified, the function will return the average attribute value of every fuel segment element in the entire dictionary. If attribute is not specified, the function call will fail. If the key value specified does not match any keys in the dictionary, the function will return a value of 0.

## **Parameters**

- groupKey (str)-
- attributeName (str) -
- nuclideSymbol (str) -
- nuclideSeries (str) -

#### Returns groupAttribute

Return type float

## GetFuelSegments (groupKey=None)

Returns a list of fuel segments associated with a specified groupkey. If no group key is specified, then all elements in the dictionary will be returned. If the specified group key does not exist, then the function will return an empty list.

```
Parameters groupKey (str) -
```

Returns fuelSegments

Return type list

#### HasGroup (key)

Checks if the specified key has a group of fuel segments associated with it.

```
Parameters key (str) –
Returns key
Return type str
```

## RemoveFuelSegment (groupKey, fuelSegment)

Removes a fuel segment from a list associated with a specified group key. If the specified group key or fuel segment do not exist, the function will fail.

#### **Parameters**

- groupKey (str)-
- fuelSegment (str) -

#### **Returns**

Return type empty

# 6.6 fuelslug module

Author: Valmor de Almeida dealmeidav@ornl.gov; vfda

Fuel slug

## 6.6.1 ATTENTION:

This container requires two Phase() containers which are by definition histories. The history is not checked. Therefore any inconsistency will be propagated forward. A fuel slug has two solid phases: cladding and fuel. The user will decide how to best use the underlying history data in the Phase() container of each phase.

VFdALib support classes

Thu Dec 15 16:18:39 EST 2016

```
class fuelslug(specs=Series([], dtype: float64), fuelPhase= **Phase()**: time unit:

s *quantities*: None *species*: None *history* #time_stamp=1 *history end* @0.0 Series([], Name: 0.0, dtype: float64), claddingPhase=

**Phase()**: time unit: s *quantities*: None *species*: None *history*

#time_stamp=1 *history end* @0.0 Series([], Name: 0.0, dtype: float64))
```

Bases: object

GetAttribute (name, phase=None, symbol=None, series=None)

Returns the value of the specified attribute. Any attribute that is specified in class construction can be retrieved using this function. The attribute may also be retrived from a specific phase, a specific nuclide OR a specific series.

## **Parameters**

- name (str) -
- phase(str) -
- symbol (str)-
- series (str) -

## Returns attribute

Return type int or float

#### GetCladdingPhase()

Returns the phase history of the cladding.

## Returns claddingPhase

Return type dataFrame

#### GetFuelPhase()

Returns the phase history of the solid fuel.

#### Returns fuelPhase

**Return type** dataFrame

#### GetSpecs()

Returns the species associated with this fuel slug.

#### Returns specs

Return type str

## ReduceCladdingVolume (dissolvedVolume)

Reduces the amount of cladding in the slug by dissolvedvolume. This will also update the dimensions of the cladding walls and end caps; volume will be taken from all sections equally such that the relative dimensions stay the same.

```
Parameters dissolvedVolume (float) -
```

## ReduceFuelVolume (dissolvedVolume)

Reduces the amount of fuel in the slug by dissolvedVolume. This will also update the dimensions of the fuel slug, mainly the thickness of each fuel layers.

```
Parameters dissolvedVolume (float) -
```

## claddingPhase

Returns the phase history of the cladding.

## Returns claddingPhase

Return type dataFrame

## fuelPhase

Returns the phase history of the solid fuel.

Returns fuelPhase

Return type dataFrame

# specs

Returns the species associated with this fuel slug.

**Returns specs** 

Return type str

# 6.7 nuclides module

Author: Valmor de Almeida dealmeidav@ornl.gov; vfda

Nuclides container. The purpose of the this container is to store and query a table of nuclides. Typically the table is filled in with data from an ORIGEN calculation or some other fission/transmutation code.

VFdALib support classes

6.7. nuclides module 56

Sat Jun 27 14:46:49 EDT 2015

 $\textbf{class} \ \ \textbf{nuclides}. \textbf{Nuclides} \ (\textit{propertyDensities=Empty DataFrame Columns: [] Index: [])}$ 

Bases: object

GetAttribute (name, symbol=None, series=None)

6.7. nuclides module 57

# **PYTHON MODULE INDEX**

a	q
actor, 41	quantity, 30
adjudication, 17 arrested, 18	r
b body, 7 C	run_city_justice, 24 run_droplet_swirl, 14 run_droplet_test, 15 run_planets, 10 run_region_justice, 10
community, 19	<b>S</b>
<pre>cortix_main, 1  d  dataplot, 8 droplet, 13</pre>	specie, 31 state, 11 stream, 38
dummy_module,8	V
f	vortex, 15
<pre>fuel_bucket, 41 fuel_bundle, 47 fuel_segment, 52 fuelsegmentsgroups, 54 fuelslug, 55</pre>	
<b>j</b> jail, 20	
m	
module, 2	
n network, 4 nuclides, 56	
р	
parole, 21 periodictable, 25 phase, 26 plot_data, 9 port, 5 prison, 22 probation, 23	

# **INDEX**

Symbols	В
del() (cortix_main.Cortix method), 1	Body (class in body), 7
eq() (port.Port method), 5	body (module), 7
init() (adjudication.Adjudication method), 17	
init() (arrested.Arrested method), 18	C
init() (community.Community method), 19	cladding_end_thickness (fuel_bucket.FuelBucket at-
init() (cortix_main.Cortix method), 1	tribute), 41
init() (droplet.Droplet method), 13	cladding_mass (fuel_bucket.FuelBucket attribute), 42
init() (jail.Jail method), 20	cladding_phase (fuel_bucket.FuelBucket attribute), 42
init() (module.Module method), 2	cladding_volume (fuel_bucket.FuelBucket attribute), 42
init() (network.Network method), 4	cladding_wall_thickness (fuel_bucket.FuelBucket at-
init() (phase.Phase method), 28	tribute), 42
init() (port.Port method), 5	claddingPhase (fuelslug.FuelSlug attribute), 56
init() (prison.Prison method), 22	ClearHistory() (phase.Phase method), 26
init() (probation.Probation method), 23	close() (cortix_main.Cortix method), 2
init() (state.State method), 11	comm (cortix_main.Cortix attribute), 1
init() (vortex.Vortex method), 15 network (module.Module attribute), 2	Community (class in community), 19
repr() (fuel_bucket.FuelBucket method), 41	community (module), 19
repr() (fuel_segment.FuelSegment method), 52	compute_velocity() (vortex.Vortex method), 16
repr() (port.Port method), 5	connect() (network.Network method), 4
_repr_() (quantity.Quantity method), 30	connect() (port.Port method), 5 Cortix (class in cortix_main), 1
str() (fuel_bucket.FuelBucket method), 41	cortix_main (module), 1
str() (fuel_segment.FuelSegment method), 52	create_plots (in module run_droplet_swirl), 14
str() (quantity.Quantity method), 31	create_plots (in module run_dropiet_swift), 14
	D
A	DataPlot (class in dataplot), 8
acceleration() (body.Body method), 7	dataplot (module), 8
Actor (class in actor), 41	draw() (network.Network method), 4
actor (module), 41	Droplet (class in droplet), 13
add_module() (network.Network method), 4	droplet (module), 13
AddGroup() (fuelsegmentsgroups.FuelSegmentsGroups	dummy_module (module), 8
method), 54	DummyModule (class in dummy_module), 8
AddQuantity() (phase.Phase method), 26	DummyModule2 (class in dummy_module), 8
AddRow() (phase.Phase method), 26	
AddSpecie() (phase.Phase method), 26	E
Adjudication (class in adjudication), 17	end_time (droplet.Droplet attribute), 13
adjudication (module), 17	end_time (in module run_city_justice), 24
Arrested (class in arrested), 18	end_time (in module run_droplet_swirl), 14
arrested (module), 18	end_time (in module run_region_justice), 11
atoms (actor.Actor attribute), 41	end_time (vortex.Vortex attribute), 15
atoms (specie.Specie attribute), 36	

F	get_cladding_volume()	(fuel_bucket.FuelBucket
flag (specie.Specie attribute), 36	method), 43	
formal_name (quantity.Quantity attribute), 31	get_cladding_wall_thickness()	(fuel_bucket.FuelBucket
formalName (quantity.Quantity attribute), 31	method), 43	
formula (actor.Actor attribute), 41	get_fresh_u235_mass()	(fuel_bucket.FuelBucket
formula (specie.Specie attribute), 36	method), 44	
formula_name (specie.Specie attribute), 36	get_fresh_U235_mass()	$(fuel\_bundle.FuelBundle$
fresh_u235_mass (fuel_bucket.FuelBucket attribute), 42	method), 49	
fresh_u235_mass (fuel_bundle.FuelBundle attribute), 48	get_fresh_u238_mass()	(fuel_bucket.FuelBucket
fresh_u238_mass (fuel_bucket.FuelBucket attribute), 42	method), 44	
fresh_u238_mass (fuel_bundle.FuelBundle attribute), 48	get_fresh_u238_mass()	$(fuel\_bundle.FuelBundle$
fresh_u_mass (fuel_bucket.FuelBucket attribute), 42	method), 49	
fresh_u_mass (fuel_bundle.FuelBundle attribute), 48	get_fresh_u_mass() (fuel_bucke	
fuel_bucket (module), 41	get_fresh_u_mass() (fuel_bund	lle.FuelBundle method),
fuel_bundle (module), 47	49	
fuel_enrichment (fuel_bucket.FuelBucket attribute), 42	get_fuel_enrichment() (fuel_buck	cket.FuelBucket method),
fuel_enrichment (fuel_bundle.FuelBundle attribute), 48	44	
fuel_mass (fuel_bucket.FuelBucket attribute), 42	get_fuel_enrichment() (fuel_bur	ndle.FuelBundle method),
fuel_mass (fuel_bundle.FuelBundle attribute), 48	50	
fuel_mass_unit (fuel_bucket.FuelBucket attribute), 42	get_fuel_mass() (fuel_bucket.Fu	
fuel_mass_unit (fuel_bundle.FuelBundle attribute), 48	get_fuel_mass() (fuel_bundle.Fu	
fuel_phase (fuel_bucket.FuelBucket attribute), 43	get_fuel_mass_unit() (fuel_buc	ket.FuelBucket method),
fuel_pin_length (fuel_bundle.FuelBundle attribute), 48	44	
fuel_pin_radius (fuel_bundle.FuelBundle attribute), 48	get_fuel_mass_unit() (fuel_bun	dle.FuelBundle method),
fuel_pin_volume (fuel_bundle.FuelBundle attribute), 48	50	
fuel_radioactivity (fuel_bucket.FuelBucket attribute), 43	get_fuel_phase() (fuel_bucket.F	
fuel_radioactivity (fuel_bundle.FuelBundle attribute), 49	get_fuel_pin_length() (fuel_bur	dle.FuelBundle method),
fuel_rod_od (fuel_bundle.FuelBundle attribute), 49	50	
fuel_segment (module), 52	get_fuel_pin_radius() (fuel_bun	dle.FuelBundle method),
fuel_volume (fuel_bucket.FuelBucket attribute), 43	50	
fuel_volume (fuel_bundle.FuelBundle attribute), 49	get_fuel_pin_volume()	$(fuel\_bundle.FuelBundle$
FuelBucket (class in fuel_bucket), 41	method), 50	
FuelBundle (class in fuel_bundle), 48	get_fuel_radioactivity()	(fuel_bucket.FuelBucket
fuelPhase (fuelslug.FuelSlug attribute), 56	method), 44	
FuelSegment (class in fuel_segment), 52	get_fuel_radioactivity()	$(fuel\_bundle.FuelBundle$
FuelSegmentsGroups (class in fuelsegmentsgroups), 54	method), 50	
fuelsegmentsgroups (module), 54	get_fuel_rod_od() (fuel_bundle.	
FuelSlug (class in fuelslug), 55	get_fuel_volume() (fuel_bucket	
fuelslug (module), 55	get_fuel_volume() (fuel_bundle	, ,
	get_gamma_pwr() (fuel_bucket.	
G	get_gamma_pwr() (fuel_bundle	
gamma_pwr (fuel_bucket.FuelBucket attribute), 43	get_gas_mass() (fuel_bundle.Fu	
gamma_pwr (fuel_bundle.FuelBundle attribute), 49	get_gas_phase() (fuel_bundle.Fi	
gas_mass (fuel_bundle.FuelBundle attribute), 49	get_gas_radioactivity()	(fuel_bundle.FuelBundle
gas_phase (fuel_bundle.FuelBundle attribute), 49	method), 51	10 10 70
gas_radioactivity (fuel_bundle.FuelBundle attribute), 49	get_geometry() (fuel_segment.F	
geometry (fuel_segment.FuelSegment attribute), 52	get_heat_pwr() (fuel_bucket.Fuel	
get_attribute() (fuel_segment.FuelSegment method), 52	get_heat_pwr() (fuel_bundle.Fu	
get_cladding_end_thickness() (fuel_bucket.FuelBucket	get_inner_slug_id() (fuel_bucke	
method), 43	get_inner_slug_od() (fuel_bucl	ket.FuelBucket method),
get_cladding_mass() (fuel_bucket.FuelBucket method),	45	EnglDundle
43	get_n_fuel_rods() (fuel_bundle.	
get_cladding_phase() (fuel_bucket.FuelBucket method),	get_n_slugs() (fuel_bucket.Fuel	
43	get_name() (fuel_bucket.FuelBu	icket memou), 45

get_name() (fuel_bundle.FuelBundle method), 51	GetQuantities() (phase.Phase method), 27	
get_name() (quantity.Quantity method), 31	GetQuantities() (stream.Stream method), 38	
get_outer_slug_id() (fuel_bucket.FuelBucket method), 45	GetQuantity() (phase.Phase method), 27	
get_outer_slug_od() (fuel_bucket.FuelBucket method),	GetQuantity() (stream.Stream method), 38	
45	GetRow() (phase.Phase method), 27	
get_port() (module.Module method), 3	GetRow() (stream.Stream method), 38	
get_quantity() (phase.Phase method), 28	GetSpecie() (phase.Phase method), 27	
get_quantity_history() (phase.Phase method), 28	GetSpecie() (stream.Stream method), 39	
get_radioactivity() (fuel_bucket.FuelBucket method), 45	GetSpecies() (phase.Phase method), 27	
get_radioactivity() (fuel_bundle.FuelBundle method), 51	GetSpecies() (stream.Stream method), 39	
get_slug_cladding_volume() (fuel_bucket.FuelBucket	GetSpecs() (fuelslug.FuelSlug method), 56	
method), 45	GetTimeStamp() (stream.Stream method), 39	
get_slug_fuel_volume() (fuel_bucket.FuelBucket	GetTimeStamps() (phase.Phase method), 27	
method), 46	GetUnit() (quantity.Quantity method), 30	
get_slug_length() (fuel_bucket.FuelBucket method), 46	GetValue() (phase.Phase method), 27	
get_slug_type() (fuel_bucket.FuelBucket method), 46	GetValue() (quantity.Quantity method), 30	
get_solid_phase() (fuel_bundle.FuelBundle method), 51	GetValue() (stream.Stream method), 39	
get_specie() (fuel_segment.FuelSegment method), 53		
get_species() (fuel_segment.FuelSegment method), 53	Н	
get_value() (phase.Phase method), 29	has_time_stamp() (phase.Phase method), 29	
GetActors() (phase.Phase method), 27	HasGroup() (fuelsegmentsgroups.FuelSegmentsGroups	
GetActors() (stream.Stream method), 38	method), 54	
GetAtoms() (specie.Specie method), 32	heat_pwr (fuel_bucket.FuelBucket attribute), 46	
GetAttribute() (fuelsegments-	heat_pwr (fuel_bundle.FuelBundle attribute), 51	
groups.FuelSegmentsGroups method), 54	p w (ruer_canaier acreament autreate), e r	
GetAttribute() (fuelslug.FuelSlug method), 55		
GetAttribute() (nuclides.Nuclides method), 57	id (port.Port attribute), 5	
GetCladdingPhase() (fuelslug.FuelSlug method), 55	initial_time (droplet.Droplet attribute), 13	
GetColumn() (phase.Phase method), 27	initial_time (vortex.Vortex attribute), 15	
GetFlag() (specie.Specie method), 32	inner_slug_id (fuel_bucket.FuelBucket attribute), 46	
GetFormalName() (quantity.Quantity method), 30	inner_slug_od (fuel_bucket.FuelBucket attribute), 46	
GetFormula() (specie.Specie method), 32	illier_stug_od (tuer_bucket.fruerbucket attiribute), 40	
GetFormulaName() (specie.Specie method), 32	J	
GetFuelPhase() (fuelslug.FuelSlug method), 56		
GetFuelSegments() (fuelsegments-	Jail (class in jail), 20	
groups.FuelSegmentsGroups method), 54	jail (module), 20	
GetMassCC() (specie.Specie method), 32	M	
GetMassCCUnit() (specie.Specie method), 33		
GetMolarCC() (specie.Specie method), 33	main() (in module run_city_justice), 24	
GetMolarCCUnit() (specie.Specie method), 33	main() (in module run_droplet_swirl), 14	
GetMolarGammaPwr() (specie.Specie method), 33	main() (in module run_planets), 10	
GetMolarGammaPwrUnit() (specie.Specie method), 33	main() (in module run_region_justice), 10	
GetMolarHeatPwr() (specie.Specie method), 33	massCC (specie.Specie attribute), 36	
GetMolarHeatPwrUnit() (specie.Specie method), 33	massCCUnit (specie.Specie attribute), 36	
GetMolarMass() (specie.Specie method), 33	max_n_modules_for_data_copy_on_root (net-	
GetMolarMassUnit() (specie.Specie method), 33	work.Network attribute), 4	
GetMolarRadioactivity() (specie.Specie method), 33	Module (class in module), 2	
GetMolarRadioactivityFractions() (specie.Specie	module (module), 2	
method), 34	module() (network.Network method), 4	
GetMolarRadioactivityUnit() (specie.Specie method), 34	molarCC (specie.Specie attribute), 36	
GetName() (specie.Specie method), 34	molarCCUnit (specie.Specie attribute), 36	
GetNAtoms() (specie.Specie method), 34	molarGammaPwr (specie.Specie attribute), 36	
GetNNuclideTypes() (specie.Specie method), 34	molarGammaPwrUnit (specie.Specie attribute), 37	
GetPhase() (specie Specie method) 34	molarHeatPwr (specie.Specie attribute), 37	

molarHeatPwrUnit (specie.Specie attribute), 37	Q
molarMass (specie.Specie attribute), 37 molarMassUnit (specie.Specie attribute), 37 molarRadioactivity (specie.Specie attribute), 37 molarRadioactivityFractions (specie.Specie attribute), 37	quantities (phase.Phase attribute), 29 Quantity (class in quantity), 30 quantity (module), 30
molarRadioactivityUnit (specie.Specie attribute), 37	R
N	radioactivity (fuel_bucket.FuelBucket attribute), 47 radioactivity (fuel_bundle.FuelBundle attribute), 52
n_droplets (in module run_droplet_swirl), 14 n_fuel_rods (fuel_bundle.FuelBundle attribute), 51 n_groups (in module run_city_justice), 24 n_groups (in module run_region_justice), 10 n_networks (network.Network attribute), 4, 5 n_slugs (fuel_bucket.FuelBucket attribute), 46 name (fuel_bucket.FuelBucket attribute), 46 name (fuel_bundle.FuelBundle attribute), 51 name (module.Module attribute), 2 name (port.Port attribute), 5 name (quantity.Quantity attribute), 31 name (specie.Specie attribute), 38 nAtoms (specie.Specie attribute), 37 Network (class in network), 4 network (cortix_main.Cortix attribute), 1, 2 network (module), 4 network (module.Module attribute), 3 nNuclideTypes (specie.Specie attribute), 38 Nuclides (class in nuclides), 57 nuclides (module), 56	rank (cortix_main.Cortix attribute), 1 recv() (module.Module method), 3 recv() (port.Port method), 5 recv_data() (dataplot.DataPlot method), 8 ReduceCladdingVolume() (fuelslug.FuelSlug method), 56 ReduceFuelVolume() (fuelslug.FuelSlug method), 56 RemoveFuelSegment() (fuelsegments- groups.FuelSegmentsGroups method), 55 ResetHistory() (phase.Phase method), 27 run() (adjudication.Adjudication method), 17 run() (arrested.Arrested method), 18 run() (body.Body method), 7 run() (community.Community method), 19 run() (cortix_main.Cortix method), 2 run() (dataplot.DataPlot method), 8 run() (dummy_module.DummyModule method), 8 run() (dummy_module.DummyModule2 method), 8 run() (jail.Jail method), 20
0	run() (module.Module method), 3 run() (network.Network method), 5
outer_slug_id (fuel_bucket.FuelBucket attribute), 46 outer_slug_od (fuel_bucket.FuelBucket attribute), 46	run() (parole.Parole method), 21 run() (plot_data.PlotData method), 9
P	run() (prison.Prison method), 22 run() (probation.Probation method), 23
Parole (class in parole), 21 parole (module), 21 periodictable (module), 25 Phase (class in phase), 26 phase (module), 26 phase (specie.Specie attribute), 38 plot() (quantity.Quantity method), 31 plot_data (module), 9	run() (state.State method), 11 run() (vortex.Vortex method), 16 run_city_justice (module), 24 run_droplet_swirl (module), 14 run_droplet_test (module), 15 run_planets (module), 10 run_region_justice (module), 10
plot_data() (dataplot.DataPlot method), 8 plot_velocity() (vortex.Vortex method), 16 plot_vortex_profile (in module run_droplet_swirl), 14 PlotData (class in plot_data), 9 Port (class in port), 5 port (module), 5 port_names_expected (module.Module attribute), 2 ports (module.Module attribute), 2 Prison (class in prison), 22	ScaleRow() (phase.Phase method), 28 send() (module.Module method), 4 send() (port.Port method), 6 set_cladding_phase() (fuel_bucket.FuelBucket method), 47 set_fuel_phase() (fuel_bucket.FuelBucket method), 47 set_fuel_pin_length() (fuel_bundle.FuelBundle method), 52
prison (module), 22 Probation (class in probation), 23 probation (module), 23	set_gas_phase() (fuel_bundle.FuelBundle method), 52 set_slug_length() (fuel_bucket.FuelBucket method), 47

set_solid_phase() (fuel_bundle.FuelBundle method), 52	time_step (droplet.Droplet attribute), 13
set_value() (phase.Phase method), 29	time_step (in module run_city_justice), 24
SetAtoms() (specie.Specie method), 34	time_step (in module run_droplet_swirl), 14
SetFlag() (specie.Specie method), 34	time_step (in module run_region_justice), 11
SetFormalName() (quantity.Quantity method), 30	time_step (vortex.Vortex attribute), 16
SetFormula() (specie.Specie method), 34	time_unit (phase.Phase attribute), 29
SetFormulaName() (specie.Specie method), 34	timeStamps (phase.Phase attribute), 29
SetMassCC() (specie.Specie method), 35	
SetMassCCUnit() (specie.Specie method), 35	U
SetMolarCC() (specie.Specie method), 35	unit (quantity.Quantity attribute), 31
SetMolarCCUnit() (specie.Specie method), 35	use_mpi (cortix_main.Cortix attribute), 1
SetMolarGammaPwr() (specie.Specie method), 35	use_mpi (in module run_city_justice), 24
SetMolarGammaPwrUnit() (specie.Specie method), 35	use_mpi (in module run_droplet_swirl), 14
SetMolarHeatPwr() (specie.Specie method), 35	use_mpi (in module run_region_justice), 11
SetMolarHeatPwrUnit() (specie.Specie method), 35	use_mpi (module.Module attribute), 2
SetMolarMass() (specie.Specie method), 35	use_mpi (port.Port attribute), 5
SetMolarMassUnit() (specie.Specie method), 35	use_multiprocessing (cortix_main.Cortix attribute), 1
SetMolarRadioactivity() (specie.Specie method), 35	use_multiprocessing (module.Module attribute), 2
SetMolarRadioactivityFractions() (specie.Specie	
method), 35	V
SetMolarRadioactivityUnit() (specie.Specie method), 35	value (quantity.Quantity attribute), 31
SetName() (quantity.Quantity method), 30	Vortex (class in vortex), 15
SetName() (specie.Specie method), 36	vortex (module), 15
SetPhase() (specie.Specie method), 36	
SetSpecieId() (phase.Phase method), 28	W
SetSpecieId() (stream.Stream method), 39	WriteHTML() (phase.Phase method), 28
SetUnit() (quantity.Quantity method), 30	wheelilivib() (phase.i hase method), 20
SetValue() (phase.Phase method), 28	
SetValue() (quantity.Quantity method), 30	
SetValue() (stream.Stream method), 39	
show_time (droplet.Droplet attribute), 13	
show_time (vortex.Vortex attribute), 16	
size (cortix_main.Cortix attribute), 2	
slug_cladding_volume (fuel_bucket.FuelBucket at-	
tribute), 47	
slug_fuel_volume (fuel_bucket.FuelBucket attribute), 47	
slug_length (fuel_bucket.FuelBucket attribute), 47	
slug_type (fuel_bucket.FuelBucket attribute), 47	
solid_phase (fuel_bundle.FuelBundle attribute), 52	
Specie (class in specie), 32	
specie (fuel_segment.FuelSegment attribute), 53	
specie (module), 31	
species (fuel_segment.FuelSegment attribute), 53	
species (phase.Phase attribute), 29	
specs (fuelslug.FuelSlug attribute), 56	
splash (cortix_main.Cortix attribute), 1	
State (class in state), 11	
state (module), 11	
state (module.Module attribute), 2	
Stream (class in stream), 38	
stream (module), 38	
Т	

time\_stamps (phase.Phase attribute), 29