# **Cortix Documentation**

Release 0.1.0

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

## **CONTENTS**

1	src		1
	1.1	cortix_main	1
	1.2	module module	1
	1.3	port module	2
2	exam	nples	4
	2.1	adjudication module	4
	2.2	arrested module	5
	2.3	community module	5
	2.4	dataplot module	6
	2.5	droplet module	6
	2.6	dummy_module module	7
	2.7	jail module	8
	2.8	parole module	9
	2.9	plot_data module	9
	2.10	prison module	10
	2.11	probation module	11
	2.12	run_droplet module	11
	2.13	run_justice module	11
	2.14	vortex module	11
2			12
3	suppo		13
	3.1	actor	13 13
		fuel_bucket	
	3.3 3.4	fuel_bundle	21 27
	3.5	fuel_segment	29
	3.6	fuelsegmentsgroups	30
	3.7	fuelslug	32
	3.8	nuclides	32 32
	3.9	periodictable	33
		phase	38
	3.10	quantity	
	3.11	specie	40
	3.12	stream	49
Py	Python Module Index		
In	dex		52

**CHAPTER** 

ONE

SRC

## 1.1 cortix main

```
class cortix_main.Cortix(use_mpi=False, splash=True)
    Bases: object
```

The main Cortix class definition: 1. Create the object 2. Add modules 3. Run the simulation

\_\_del\_\_()

Note: 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.

### add\_module (m)

Add a module to the Cortix object

m: An instance of a class that inherits from the Module base class

```
draw_network (file_name='network.png', dpi=220)
```

Draws the networkx Module network graph using matplotlib

file\_name: The resulting network diagram output file name dpi: dpi used for generating the network image

### get\_network()

Constructs and returns a networkx graph representation of the module network.

run()

Run the Cortix simulation with MPI if available o.w. fallback to multiprocessing

### 1.2 module module

### class module.Module

Bases: object

The representation of a Cortix module. This class is to be inherited by every Cortix module. It provides facilities for creating modules within the Cortix network. In addition Cortix will map one object of this class to a Python multiprocessing child process, or one executable object of this class to one MPI process.

```
___init___()
```

### port\_names\_expected

*list(str) or 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.

```
connect (port_name, other_port)
```

A simpler interface to create module connectivity. Connect the module port with *my\_port\_name* to a given *other\_port*.

```
get port(name)
```

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

### recv (port)

Receive data from a given port

```
run (state_comm=None, idx_comm=None)
```

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

### **Notes**

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

#### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

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

Send data through a given port.

## 1.3 port module

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

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.

```
__eq__ (other)
Ports are the same if their names are the same
__repr__ ()
Port name representation

connect (port)
Connect the port to another port
```

port: A Port object that represents the port to connect to.

recv()

Returns the data received from the connected port.

1.3. port module 2

send(data, tag=None)

Send data to the connected port. If the sending port is not connected do nothing.

Parameters data(any pickle-able data)-

Returns

Return type None

1.3. port module 3

**CHAPTER** 

**TWO** 

### **EXAMPLES**

## 2.1 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 (state\_comm=None, idx\_comm=None)

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

### Notes

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

#### **Parameters**

- **state\_comm** (*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*.
- idx comm (int) Index of the state in the communication queue.

## 2.2 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 (state\_comm=None, idx\_comm=None)

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

#### **Notes**

For now, add this command: *state\_comm.put((idx\_comm,self.state))* to the bottom of your method. If you are not using *self.state*, then the command can be issued anywhere in the body of the function.

### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

## 2.3 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, maturity_rate=0.00011574074074074075, offender_pool_size=0.0)
```

2.2. arrested module 5

### **Parameters**

- **n\_groups** (*int*) Number of groups in the population.
- maturity\_rate (float) Rate of individuals reaching the adult age (SI) unit. Default: 10 per day.
- **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.

run (state comm=None, idx comm=None)

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

#### **Notes**

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

#### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

## 2.4 dataplot module

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

plot_data()

recv_data(port)

   Keep listening on the port and receiving data.

run (state_comm=None, idx_comm=None)
   Spawn a thread to handle each port connection.
```

## 2.5 droplet module

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

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

**Note:** *external-flow*: this is a *port* that exchanges data with any other module that provides information about the flow outside the droplet.

visualization: this is a port that sends data to a visualization module.

```
rhs_fn (u_vec, t, params)
```

**run** (*state comm=None*, *idx comm=None*)

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

#### **Notes**

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

step(time=0.0)

ODE IVP problem: Given the initial data at t=0,  $(u_1(0),u_2(0),u_3(0))=(x_0,x_1,x_2)$ ,  $(u_4(0),u_5(0),u_6(0))=(v_0,v_1,v_2)=(\dot{u}_1(0),\dot{u}_2(0),\dot{u}_3(0))$ , solve  $\frac{\mathrm{d}u}{\mathrm{d}t}=f(u)$  in the interval  $0\leq t\leq t_f$ . When  $u_3(t)$  is negative, bounce the droplet to a random height between 0 and  $1.0\,x_0$  with no velocity, and continue the time integration until  $t=t_f$ .

**Parameters** time (float) – Time in the droplet unit of time (seconds).

Returns

Return type None

## 2.6 dummy\_module module

```
class dummy_module.DummyModule
```

Bases: cortix.src.module.Module

run (state\_comm=None, idx\_comm=None)

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

### **Notes**

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

### class dummy\_module.DummyModule2

Bases: cortix.src.module.Module

run (state\_comm=None, idx\_comm=None)

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

#### **Notes**

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

#### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

## 2.7 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 (state\_comm=None, idx\_comm=None)

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

### **Notes**

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

#### **Parameters**

2.7. jail module 8

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

## 2.8 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*.

```
compute_outflow_rates (time, name)
```

```
run (state comm=None, idx comm=None)
```

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

### **Notes**

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

## 2.9 plot\_data module

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

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

### Notes

2.8. parole module 9

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

#### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

## 2.10 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 (state\_comm=None, idx\_comm=None)

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

#### **Notes**

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

## 2.11 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 (state\_comm=None, idx\_comm=None)

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

#### **Notes**

For now, add this command: *state\_comm.put((idx\_comm,self.state))* to the bottom of your method. If you are not using *self.state*, then the command can be issued anywhere in the body of the function.

### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

## 2.12 run\_droplet module

## 2.13 run\_justice module

### 2.14 vortex module

class vortex.Vortex

Bases: cortix.src.module.Module

Vortex module used to model fluid flow using Cortix.

**Note:** Any *port* name and any number of ports are allowed.

### compute\_velocity (time, position)

Compute the vortex velocity at the given external position using a vortex flow model

#### **Parameters**

- time (float) Time in SI unit.
- position (numpy.ndarray (3)) Spatial position in SI unit.

### Returns vortex\_velocity

**Return type** numpy.ndarray(3)

### plot\_velocity(time=None)

Plot the vortex velocity as a function of height.

```
run (state_comm=None, idx_comm=None)
```

Run method with an option to pass data back to the parent process when running Python multiprocessing mode.

### Notes

For now, add this command:  $state\_comm.put((idx\_comm,self.state))$  to the bottom of your method. If you are not using self.state, then the command can be issued anywhere in the body of the function.

### **Parameters**

- **state\_comm** (*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*.
- idx\_comm (int) Index of the state in the communication queue.

2.14. vortex module

**CHAPTER** 

**THREE** 

### **SUPPORT**

### 3.1 actor

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.

Parameters empty -
Returns atoms
Return type list(str)

formula
Returns the formula of the chemical in question.

Parameters empty -
Returns formula
Return type str
```

## 3.2 fuel\_bucket

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

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

### 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.

Parameters empty -

Returns cladding\_end\_thickness

Return type cm

### cladding\_mass

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

Parameters empty -

Returns cladding\_mass

Return type float

### cladding\_phase

Returns the phase history of the cladding.

Parameters empty –

Returns cladding\_phase

Return type dataFrame

### cladding\_volume

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

Parameters empty –

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.

Parameters empty -

Returns cladding\_wall\_thickness

**Return type** float

### ${\tt fresh\_u235\_mass}$

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

Parameters empty -

Returns fresh\_u235\_mass

Return type float

### fresh\_u238\_mass

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

Parameters empty -

Returns fresh\_u238\_mass

Return type float

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

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

3.2. fuel bucket

```
Parameters empty -
         Returns fresh_u_mass
         Return type float
fuel enrichment
     Returns the enrichment of the fuel slugs in the bucket, in %.
         Parameters empty -
         Returns fuel_enrichment
         Return type float
fuel mass
     Returns the total mass of fuel in the solid phase in the bucket.
         Parameters empty -
         Returns fuel_mass
         Return type float
fuel_mass_unit
     Returns the unit that is used to measure the mass of fuel in the bucket.
         Parameters empty -
         Returns fuel mass unit
         Return type str
fuel_phase
     Returns the phase history of the fuel.
         Parameters empty -
         Returns fuel_phase
         Return type pandas.core.frame.DataFrame
fuel_radioactivity
     Returns the total radioactivity of the solid phase fuel, in units of curies.
         Parameters empty -
         Returns fuel_radioactivity
         Return type float
fuel volume
     Returns the total volume of fuel in the entire bucket, in cm<sup>3</sup>.
         Parameters empty -
         Returns fuel_volume
         Return type float
gamma_pwr
     Returns the amount of gamma radiation given off by the fuel bucket, in units of watts.
         Parameters empty -
         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.
         Parameters empty -
         Returns cladding_end_thickness
         Return type cm
get_cladding_mass()
     Returns the total mass of cladding material in the bucket, in grams.
         Parameters empty -
         Returns cladding_mass
         Return type float
get_cladding_phase()
     Returns the phase history of the cladding.
         Parameters empty -
         Returns cladding phase
         Return type dataFrame
get_cladding_volume()
     Returns the total volume of cladding in the bucket, in cm<sup>3</sup>.
         Parameters empty –
         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.
         Parameters empty -
         Returns cladding wall thickness
         Return type float
get_fresh_u235_mass()
```

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

```
Parameters empty –
```

Returns fresh\_u235\_mass

Return type float

```
get_fresh_u238_mass()
```

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

Parameters empty -

Returns fresh\_u238\_mass

Return type float

```
get_fresh_u_mass()
```

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

3.2. fuel bucket 16

```
Parameters empty -
         Returns fresh_u_mass
         Return type float
get_fuel_enrichment()
    Returns the enrichment of the fuel slugs in the bucket, in %.
         Parameters empty –
         Returns fuel_enrichment
         Return type float
get_fuel_mass()
     Returns the total mass of fuel in the solid phase in the bucket.
         Parameters empty -
         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.
         Parameters empty -
         Returns fuel mass unit
         Return type str
get_fuel_phase()
     Returns the phase history of the fuel.
         Parameters empty -
         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.
         Parameters empty -
         Returns fuel_radioactivity
         Return type float
get fuel volume()
     Returns the total volume of fuel in the entire bucket, in cm<sup>3</sup>.
         Parameters empty -
         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.
         Parameters empty -
         Returns gamma_pwr
         Return type float
```

```
get_heat_pwr()
     Returns the total amount of heat generated by the bucket, in units of watts.
         Parameters empty -
         Returns heat_pwr
         Return type float
get_inner_slug_id()
     Returns the inner diameter of the inner section of fuel, in cm.
         Parameters empty -
         Returns inner_slug_id
         Return type float
get_inner_slug_od()
     Returns the outer diameter of the inner section of fuel, in cm.
         Parameters empty -
         Returns inner_slug_od
         Return type float
get_n_slugs()
     Returns the number of fuel slugs in the bucket.
         Parameters empty -
         Returns n_slugs
         Return type int
get_name()
     Returns the name of the fuel bucket.
         Parameters empty -
         Returns name
         Return type str
get_outer_slug_id()
    Returns the inner diameter of the outer section of fuel, in cm.
         Parameters empty -
         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.
         Parameters empty -
         Returns outer_slug_od
         Return type float
get_radioactivity()
```

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

```
Parameters empty -
         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>.
         Parameters empty –
         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>.
         Parameters empty -
         Returns slug_fuel_volume
         Return type float
get_slug_length()
     Returns the length of each slug in the fuel bucket.
         Parameters empty -
         Returns slug length
         Return type float
get_slug_type()
     Returns the type of slugs being stored in the bucket (inner slug or outer slug).
         Parameters empty -
         Returns slug_type
         Return type str
heat_pwr
     Returns the total amount of heat generated by the bucket, in units of watts.
         Parameters empty -
         Returns heat_pwr
         Return type float
inner slug id
     Returns the inner diameter of the inner section of fuel, in cm.
         Parameters empty -
         Returns inner_slug_id
         Return type float
inner_slug_od
     Returns the outer diameter of the inner section of fuel, in cm.
         Parameters empty -
         Returns inner_slug_od
         Return type float
```

#### n slugs

Returns the number of fuel slugs in the bucket.

Parameters empty -

Returns n\_slugs

**Return type** int

#### name

Returns the name of the fuel bucket.

Parameters empty -

Returns name

Return type str

### outer\_slug\_id

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

Parameters empty -

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.

Parameters empty -

Returns outer\_slug\_od

Return type float

### radioactivity

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

Parameters empty -

Returns radioactivity

Return type float

### set\_cladding\_phase (phase)

Set's the phase history to specific values.

Parameters phase (dataFrame) -

Returns

Return type empty

### $\mathtt{set\_fuel\_phase}\ (phase)$

Sets the current fuel phase to a specified phase value.

Parameters phase (dataFrame) -

Returns

Return type empty

### $set\_slug\_length(x)$

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

3.2. fuel bucket 20

```
Parameters x (float) -
         Returns
         Return type empty
slug_cladding_volume
     Returns the volume of cladding present in a single fuel slug, in cm<sup>3</sup>.
         Parameters empty –
         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>.

Parameters empty -

Returns slug\_fuel\_volume

Return type float

### slug\_length

Returns the length of each slug in the fuel bucket.

Parameters empty -Returns slug length Return type float

### slug\_type

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

Parameters empty -Returns slug\_type Return type str

## 3.3 fuel bundle

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

```
class fuel_bundle.FuelBundle(specs=Empty DataFrame Columns: [] Index: [])
    Bases: object
```

### fresh u235 mass

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

Parameters empty -

```
Returns fresh_u235_mass
```

Return type float

### fresh\_u238\_mass

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

Parameters empty -

Returns fresh\_u238\_mass

Return type float

### fresh\_u\_mass

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

Parameters empty -

Returns fresh\_u\_mass

Return type float

#### fuel enrichment

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

Parameters empty -

Returns fuel\_enrichment

Return type float

### fuel mass

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

Parameters empty -

Returns fuel\_mass

Return type float

### fuel\_mass\_unit

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

Parameters empty -

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.

Parameters empty -

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>.

Parameters empty -

Returns fuel\_pin\_volume

### Return type float

### fuel\_radioactivity

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

Parameters empty -

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.

Parameters empty -

Returns fuel\_rod\_od

Return type float

### fuel\_volume

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

Parameters empty -

Returns fuel\_volume

Return type float

#### gamma pwr

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

Parameters empty -

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.

Parameters empty -

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.

Parameters empty -

Returns gas\_radioactivity

Return type float

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

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

Parameters empty -

Returns fresh\_u235\_mass

Return type float

```
get_fresh_u238_mass()
     Returns the amount of uranium-238 in the bucket, in grams.
         Parameters empty -
         Returns fresh u238 mass
         Return type float
get_fresh_u_mass()
     Returns the amount of uranium in the bundle, in grams.
         Parameters empty -
         Returns fresh_u_mass
         Return type float
get_fuel_enrichment()
     Returns the enrichment of the fuel pins in the bundle, in %.
         Parameters empty -
         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.
         Parameters empty -
         Returns fuel_mass
         Return type float
get_fuel_mass_unit()
     Returns the unit used to measure the mass of fuel in the bundle.
         Parameters empty -
         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.
         Parameters empty -
         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>.
         Parameters empty -
         Returns fuel_pin_volume
```

3.3. fuel bundle 24

Return type float

```
get_fuel_radioactivity()
     Returns the total radioactivity of the fuel in the solid phase in the fuel bundle.
         Parameters empty -
         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.
         Parameters empty -
         Returns fuel_rod_od
         Return type float
get_fuel_volume()
     Returns the total volume of fuel in the bundle, in cm<sup>3</sup>.
         Parameters empty -
         Returns fuel_volume
         Return type float
get_gamma_pwr()
     Returns the total amount of gamma radiation given by the fuel bundle, in watts.
         Parameters empty -
         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.
         Parameters empty -
         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.
         Parameters empty -
         Returns gas radioactivity
         Return type float
get_heat_pwr()
     Returns the total amount of heat produced by the fuel bundle, in watts.
         Parameters empty –
         Returns heat_pwr
         Return type float
get_n_fuel_rods()
     Returns the number of fuel rods in the bundle.
```

```
Parameters empty -
         Returns n_fuel_rods
         Return type int
get_name()
     Returns the name of the fuel bundle.
         Parameters empty -
         Returns name
         Return type str
get_radioactivity()
     Returns the total radioactivity of the fuel bundle, in curies.
         Parameters empty -
         Returns raduioactivity
         Return type float
get_solid_phase()
     Returns the solid phase history associated with this fuel bundle.
         Parameters empty -
         Returns solidPhase
         Return type dataFrame
heat_pwr
     Returns the total amount of heat produced by the fuel bundle, in watts.
         Parameters empty -
         Returns heat_pwr
         Return type float
n_fuel_rods
     Returns the number of fuel rods in the bundle.
         Parameters empty -
         Returns n_fuel_rods
         Return type int
name
     Returns the name of the fuel bundle.
         Parameters empty -
         Returns name
         Return type str
radioactivity
     Returns the total radioactivity of the fuel bundle, in curies.
         Parameters empty -
         Returns raduioactivity
         Return type float
```

27

```
set_fuel_pin_length(x)
          Sets the length of all fuel pins in the bundle to x.
              Parameters empty -
              Returns x
              Return type float
     set_gas_phase(phase)
          Sets the gas phase history of the fuel equal to phase.
              Parameters phase (dataFrame) -
              Returns
              Return type empty
     set_solid_phase(phase)
          Sets the solid phase history of the fuel equal to phase.
              Parameters phase (dataFrame) -
              Returns
              Return type empty
     solid phase
          Returns the solid phase history associated with this fuel bundle.
              Parameters empty -
              Returns solidPhase
              Return type dataFrame
3.4 fuel_segment
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.
              Parameters empty -
              Returns s
              Return type str
     __str__()
          Used to print the geometry of the fuel segment and the species that it consists of.
```

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

Parameters empty -

Parameters Empty -

Returns s

geometry

3.4. fuel segment

Return type str

### 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).

Parameters Empty -

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

#### qet species()

Returns the species object which describes the composition of the fuel bundle. The species encapsulates all chemical species present in the fuel bundle. :param None:

### 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

3.4. fuel segment 28

### species

Returns the species object which describes the composition of the fuel bundle. The species encapsulates all chemical species present in the fuel bundle. :param None:

### **Returns species**

Return type object

## 3.5 fuelsegmentsgroups

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

Fuel segment

VFdALib support classes

Sat Jun 27 14:46:49 EDT 2015

```
\textbf{class} \ \texttt{fuelsegmentsGroups}. \textbf{\textit{FuelSegmentsGroups}} \ (\textit{\textit{key=None}}, \textit{\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) -

#### Returns

Return type empty

**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
```

### ${\tt RemoveFuelSegment} \ (\textit{groupKey}, \textit{fuelSegment})$

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

## 3.6 fuelslug

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

Fuel slug

### 3.6.1 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.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**

3.6. fuelslug 30

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

#### Returns attribute

Return type int or float

### GetCladdingPhase()

Returns the phase history of the cladding.

Parameters empty -

Returns claddingPhase

Return type dataFrame

### GetFuelPhase()

Returns the phase history of the solid fuel.

Parameters empty -

Returns fuelPhase

Return type dataFrame

### GetSpecs()

Returns the species associated with this fuel slug.

Parameters empty -

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) -

Returns

Return type empty

### ${\tt ReduceFuelVolume}\ (\textit{dissolvedVolume})$

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) -

Returns

Return type empty

### claddingPhase

Returns the phase history of the cladding.

Parameters empty -

Returns claddingPhase

Return type dataFrame

3.6. fuelslug 31

### fuelPhase

Returns the phase history of the solid fuel.

Parameters empty -

Returns fuelPhase

**Return type** dataFrame

### specs

Returns the species associated with this fuel slug.

Parameters empty -

**Returns specs** 

Return type str

### 3.7 nuclides

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

Sat Jun 27 14:46:49 EDT 2015

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

## 3.8 periodictable

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

### 3.8.1 Requirements

• CPython 2.7 or 3.4

#### References

- 1. http://physics.nist.gov/PhysRefData/Compositions/
- 2. http://physics.nist.gov/PhysRefData/IonEnergy/tblNew.html

3.7. nuclides 32

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

## 3.9 phase

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.

### 3.9.1 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) -

3.9. phase 33

### **Returns**

Return type empty

### 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)-
```

### Returns

Return type empty

### 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) -
```

Returns

Return type empty

### 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) -
```

Returns

Return type empty

### GetActors()

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

```
Parameters empty -
```

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.

Parameters empty -

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) -
```

3.9. phase 34

#### Returns

```
Return type empty
```

## 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.

Parameters empty -

**Returns species** 

Return type list

#### GetTimeStamps()

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

Parameters empty -

**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) -

## Returns

Return type empty

## 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)-

## Returns

3.9. phase 35

#### **Return type** empty

#### SetSpecieId (name, val)

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

#### **Parameters**

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

#### Returns

Return type empty

**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)-

#### Returns

Return type empty

#### WriteHTML(fileName)

Convert the *Phase* container into an HTML file.

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

Returns

Return type empty

```
__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.

3.9. phase 36

```
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.
         Parameters empty -
         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.
         Parameters empty -
         Returns species
         Return type list
timeStamps
     Returns a list of all the time stamps in the phase history.
         Parameters empty -
         Returns timeStamps
         Return type list
time stamps
     Get all time stamps in the index of the data frame.
         Parameters empty -
         Returns time_stamps
         Return type list
time unit
     Returns the time unit of the Phase.
         Parameters empty -
```

3.9. phase 37

Returns time unit

## Return type str

Return type empty

# 3.10 quantity

```
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.
               Parameters empty -
               Returns formalName
               Return type str
     GetUnit()
           Returns the units of the quantity.
               Parameters empty -
               Returns unit
               Return type str
     GetValue()
           Gets the numerical value of the quantity.
               Parameters empty -
               Returns value
               Return type any type
     {\tt SetFormalName}\ (fn)
           Sets the formal name of the property to fn.
               Parameters fn (str) -
               Returns
               Return type empty
     SetName(n)
           Sets the name of the quantity in question to n.
               Parameters n (str) -
               Returns
```

3.10. quantity 38

```
SetUnit(f)
     Sets the units of the quantity to f (for example, density would be in units of g/cc.
         Parameters f(str)-
         Returns
         Return type empty
SetValue (v)
    Sets the numerical value of the quantity to v.
         Parameters v(float)-
         Returns
         Return type empty
__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.
         Parameters empty -
         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.
         Parameters empty -
         Returns s
         Return type str
formalName
    Returns the formal name of the quantity.
         Parameters empty -
         Returns formalName
         Return type str
formal_name
     Returns the formal name of the quantity.
         Parameters empty -
         Returns formalName
         Return type str
get_name()
    Returns the name of the quantity.
         Parameters empty -
         Returns name
         Return type str
name
     Returns the name of the quantity.
```

3.10. quantity 39

Parameters empty -

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. Thi 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.

Parameters empty -

Returns unit

Return type str

#### value

Gets the numerical value of the quantity.

Parameters empty -

Returns value

**Return type** any type

## 3.11 specie

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

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
```

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.

Parameters empty -

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".

Parameters empty -

Returns self.\_\_formula\_name

Return type str

#### GetMassCC()

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

Parameters empty -

Returns massCC

Return type float

## GetMassCCUnit()

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

Parameters empty -

Returns massCCUnit

Return type str

## GetMolarCC()

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

Parameters empty -

Returns molarCC

Return type float

#### GetMolarCCUnit()

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

Parameters empty -

Returns molarCCUnit

Return type str

#### GetMolarGammaPwr()

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

Parameters empty -

Returns molarGammaPwr

Return type float

## GetMolarGammaPwrUnit()

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

Parameters empty -

Returns molarGammaPwrUnit

Return type str

#### GetMolarHeatPwr()

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

Parameters empty -

Returns molarHeatPwr

Return type float

## GetMolarHeatPwrUnit()

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

Parameters empty -

Returns molarHeatPwrUnit

Return type str

## GetMolarMass()

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

Parameters empty -

Returns molarMass

Return type float

## GetMolarMassUnit()

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

Parameters empty -

Returns molarMassUnit

Return type str

#### GetMolarRadioactivity()

Returns the numerical value for molar radioactivity of the species.

Parameters empty -

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.

Parameters empty -

Returns molarRadioactivityFractions

**Return type** list

## GetMolarRadioactivityUnit()

Returns the unit used to measure molar radioactivity.

Parameters empty -

Returns molarRadioactivityUnit

Return type str

#### GetNAtoms()

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

Parameters empty -

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.

Parameters empty -

Returns nNuclideTypes

Return type int

#### GetName()

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

Parameters empty -

Returns name

Return type str

#### GetPhase()

Returns the phase history of the species.

Parameters empty -

Returns phase

Return type dataFrame

SetAtoms (atoms)

#### SetFlag(f)

Sets the flag associated with the species to f.

Parameters f(str)-

**Returns** 

## Return type empty

```
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) -
```

Returns

Return type empty

## SetFormulaName(f)

Sets the formulaic name to f.

Parameters empty -

Returns

- empty
- self.\_\_formula\_name (str)

#### SetMassCC(v)

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

Parameters v(float)-

Returns

Return type empty

#### SetMassCCUnit(v)

Sets the units used to measure mass density to v.

Parameters v(str) -

Returns

Return type empty

## SetMolarCC(v)

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

Parameters v(float)-

Returns

Return type empty

## SetMolarCCUnit(v)

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

Parameters v(str)-

Returns

Return type empty

#### SetMolarGammaPwr(v)

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

Parameters v(float)-

Returns

Return type empty

#### SetMolarGammaPwrUnit (v)

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

Parameters v(str)-

Returns

Return type empty

#### SetMolarHeatPwr(v)

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

Parameters v(float)-

**Returns** 

Return type empty

## ${\tt SetMolarHeatPwrUnit}\;(v)$

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

Parameters v (str) -

Returns

Return type empty

#### SetMolarMass(v)

Sets the molar mass of the species equal to v.

Parameters v(float) -

Returns

Return type empty

## SetMolarMassUnit(v)

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

Parameters v(str)-

**Returns** 

**Return type** empty

## SetMolarRadioactivity(v)

Sets the molar radioactivity of the species equal to v.

Parameters v(float)-

Returns

Return type empty

## 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) -

Returns

Return type empty

## SetMolarRadioactivityUnit(v)

Sets the unit used to measure molar radioactivity to v.

```
Parameters v(str) -
         Returns
         Return type empty
SetName (n)
     Sets the empirical name of the species to n.
         Parameters n (str) -
         Returns
         Return type empty
SetPhase (p)
     Sets the phase history to p.
         Parameters p (dataFrame) -
         Returns
         Return type empty
atoms
flag
     Returns the flag associated with the species.
         Parameters empty -
         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".
         Parameters empty –
         Returns self.__formula_name
         Return type str
massCC
     Returns the numerical value of the mass density of the species (mass/volume).
         Parameters empty -
         Returns massCC
         Return type float
massCCUnit
     Returns the unit used to measure the mass density of the species.
         Parameters empty -
         Returns massCCUnit
         Return type str
```

#### molarCC

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

Parameters empty -

Returns molarCC

Return type float

#### molarCCUnit

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

Parameters empty -

Returns molarCCUnit

Return type str

## molarGammaPwr

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

Parameters empty -

Returns molarGammaPwr

Return type float

#### molarGammaPwrUnit

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

Parameters empty -

Returns molarGammaPwrUnit

Return type str

## molarHeatPwr

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

Parameters empty -

Returns molarHeatPwr

Return type float

## molarHeatPwrUnit

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

Parameters empty -

Returns molarHeatPwrUnit

Return type str

#### molarMass

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

Parameters empty -

Returns molarMass

Return type float

#### molarMassUnit

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

Parameters empty -

Returns molarMassUnit

#### Return type str

#### molarRadioactivity

Returns the numerical value for molar radioactivity of the species.

Parameters empty -

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.

Parameters empty -

Returns molarRadioactivityFractions

Return type list

## molarRadioactivityUnit

Returns the unit used to measure molar radioactivity.

Parameters empty -

Returns molarRadioactivityUnit

Return type str

#### nAtoms

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

Parameters empty -

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.

Parameters empty -

Returns nNuclideTypes

Return type int

#### name

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

Parameters empty -

Returns name

Return type str

#### phase

Returns the phase history of the species.

Parameters empty -

Returns phase

Return type dataFrame

## 3.12 stream

```
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.
               Parameters empty -
               Returns list(self.stream.columns)
               Return type list
     GetQuantities()
          Returns all the quantities given by the stream.
               Parameters empty -
               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
     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
               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.
               Parameters empty -
```

3.12. stream 49

## Returns self.species

**Return type** list

## GetTimeStamp()

Returns the time stamp of the stream.

#### Parameters empty -

## 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) -

#### Returns

Return type empty

## **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**

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

## Returns

Return type empty

3.12. stream 50

# **PYTHON MODULE INDEX**

a	r
actor, 13	run_droplet,11
adjudication, 4	run_justice,11
arrested, 5	S
C	specie, 40
community,5	stream,49
cortix_main, 1	V
d	vortex, 11
dataplot, 6	
droplet, 6	
dummy_module,7	
f	
fuel_bucket, 13	
fuel_bundle, 21	
<pre>fuel_segment, 27 fuelsegmentsgroups, 29</pre>	
fuelslug, 30	
i	
jail,8	
Jall, o	
m	
module, 1	
n	
nuclides, 32	
p	
parole, 9	
periodictable, 32	
phase, 33	
plot_data, 9 port, 2	
prison, 10	
probation, 11	
q	
quantity, 38	

# **INDEX**

Symbols	cladding_wall_thickness (fuel_bucket.FuelBucket at-
del() (cortix_main.Cortix method), 1	tribute), 14
eq() (port.Port method), 2	claddingPhase (fuelslug.FuelSlug attribute), 31
init() (adjudication.Adjudication method), 4	ClearHistory() (phase.Phase method), 34
init() (arrested.Arrested method), 5	Community (class in community), 5
init() (community.Community method), 5	community (module), 5
init() (jail.Jail method), 8	compute_outflow_rates() (parole.Parole method), 9
init() (module.Module method), 1	compute_velocity() (vortex.Vortex method), 11
init() (phase.Phase method), 36	connect() (module.Module method), 2
init() (prison.Prison method), 10	connect() (port.Port method), 2
init() (probation.Probation method), 11	Cortix (class in cortix_main), 1
repr() (fuel_bucket.FuelBucket method), 13	cortix_main (module), 1
repr() (fuel_segment.FuelSegment method), 27	D
repr() (port.Port method), 2	_
repr() (quantity.Quantity method), 39	DataPlot (class in dataplot), 6
str() (fuel_bucket.FuelBucket method), 13	dataplot (module), 6
str() (fuel_segment.FuelSegment method), 27	draw_network() (cortix_main.Cortix method), 1
str() (quantity.Quantity method), 39	Droplet (class in droplet), 6
Α	droplet (module), 6
A	dummy_module (module), 7
Actor (class in actor), 13	DummyModule (class in dummy_module), 7
actor (module), 13	DummyModule2 (class in dummy_module), 7
add_module() (cortix_main.Cortix method), 1	F
AddGroup() (fuelsegmentsgroups.FuelSegmentsGroups	
method), 29	flag (specie.Specie attribute), 46
AddQuantity() (phase.Phase method), 33	formal_name (quantity.Quantity attribute), 39
AddRow() (phase.Phase method), 34	formalName (quantity.Quantity attribute), 39
AddSpecie() (phase.Phase method), 34	formula (actor.Actor attribute), 13
Adjudication (class in adjudication), 4	formula (specie.Specie attribute), 46
adjudication (module), 4	formula_name (specie.Specie attribute), 46
Arrested (class in arrested), 5	fresh_u235_mass (fuel_bucket.FuelBucket attribute), 14
arrested (module), 5	fresh_u235_mass (fuel_bundle.FuelBundle attribute), 21
atoms (actor.Actor attribute), 13	fresh_u238_mass (fuel_bucket.FuelBucket attribute), 14
atoms (specie.Specie attribute), 46	fresh_u238_mass (fuel_bundle.FuelBundle attribute), 22
•	fresh_u_mass (fuel_bucket.FuelBucket attribute), 14
C	fresh_u_mass (fuel_bundle.FuelBundle attribute), 22
cladding_end_thickness (fuel_bucket.FuelBucket at-	fuel_bucket (module), 13
tribute), 13	fuel_bundle (module), 21
cladding_mass (fuel_bucket.FuelBucket attribute), 14	fuel_enrichment (fuel_bucket.FuelBucket attribute), 15
cladding_phase (fuel_bucket.FuelBucket attribute), 14	fuel_enrichment (fuel_bundle.FuelBundle attribute), 22
cladding_volume (fuel_bucket.FuelBucket attribute), 14	fuel_mass (fuel_bucket.FuelBucket attribute), 15
2_ \ \ _	fuel_mass (fuel_bundle.FuelBundle attribute), 22

uel_mass_unit (fuel_bucket.FuelBucket attribute), 15 uel_mass_unit (fuel_bundle.FuelBundle attribute), 22 uel_phase (fuel_bucket FuelBucket attribute), 15		get_fuel_mass() (fuel_bucket.F get_fuel_mass() (fuel_bundle.F get_fuel_mass_unit() (fuel_bud	uelBundle method), 24
fuel_phase (fuel_bucket.FuelBucket attribute), 15 fuel_pin_length (fuel_bundle.FuelBundle attribute), 22		17	Ret.FuelDucket method),
fuel_pin_radius (fuel_bundle.Fu		get_fuel_mass_unit() (fuel_bur	ndle.FuelBundle method).
fuel_pin_volume (fuel_bundle.FuelBundle attribute), 22		24	,
fuel_radioactivity (fuel_bucket.l		get_fuel_phase() (fuel_bucket.I	FuelBucket method), 17
fuel_radioactivity (fuel_bundle.)		get_fuel_pin_length() (fuel_but	
fuel_rod_od (fuel_bundle.FuelB		24	,
fuel_segment (module), 27		get_fuel_pin_radius() (fuel_but	ndle.FuelBundle method),
fuel_volume (fuel_bucket.FuelE	Bucket attribute), 15	24	
fuel_volume (fuel_bundle.FuelBundle attribute), 23		get_fuel_pin_volume()	(fuel_bundle.FuelBundle
FuelBucket (class in fuel_bucke	et), 13	method), 24	
FuelBundle (class in fuel_bundl	e), 21	get_fuel_radioactivity()	(fuel_bucket.FuelBucket
fuelPhase (fuelslug.FuelSlug att	ribute), 31	method), 17	
FuelSegment (class in fuel_segn	ment), 27	get_fuel_radioactivity()	(fuel_bundle.FuelBundle
FuelSegmentsGroups (class in f	uelsegmentsgroups), 29	method), 24	
fuelsegmentsgroups (module), 2	29	get_fuel_rod_od() (fuel_bundle	
FuelSlug (class in fuelslug), 30		get_fuel_volume() (fuel_bucket	t.FuelBucket method), 17
fuelslug (module), 30		get_fuel_volume() (fuel_bundle	e.FuelBundle method), 25
		get_gamma_pwr() (fuel_bucket	t.FuelBucket method), 17
G		get_gamma_pwr() (fuel_bundle	
gamma_pwr (fuel_bucket.FuelB	Bucket attribute), 15	get_gas_mass() (fuel_bundle.Fu	
gamma_pwr (fuel_bundle.FuelBundle attribute), 23		get_gas_phase() (fuel_bundle.F	
gas_mass (fuel_bundle.FuelBundle attribute), 23		get_gas_radioactivity()	(fuel_bundle.FuelBundle
gas_phase (fuel_bundle.FuelBu	ndle attribute), 23	method), 25	
gas_radioactivity (fuel_bundle.F	FuelBundle attribute), 23	get_geometry() (fuel_segment.)	_
geometry (fuel_segment.FuelSe	gment attribute), 27	get_heat_pwr() (fuel_bucket.Fu	
get_attribute() (fuel_segment.Fu	ielSegment method), 28	get_heat_pwr() (fuel_bundle.Fu	
get_cladding_end_thickness()	(fuel_bucket.FuelBucket	get_inner_slug_id() (fuel_bucke	
method), 15		get_inner_slug_od() (fuel_buc	ket.FuelBucket method),
get_cladding_mass() (fuel_buc	ket.FuelBucket method),	18	E 1D 11 (1.1) 27
16		get_n_fuel_rods() (fuel_bundle	
get_cladding_phase() (fuel_buc	ket.FuelBucket method),	get_n_slugs() (fuel_bucket.Fue	
16		get_name() (fuel_bucket.FuelB	
get_cladding_volume()	(fuel_bucket.FuelBucket	get_name() (fuel_bundle.FuelB	, ,
method), 16		get_name() (quantity.Quantity i	
get_cladding_wall_thickness()	(fuel_bucket.FuelBucket	<pre>get_network() (cortix_main.Co get_outer_slug_id() (fuel_bucket</pre>	
method), 16	(6 1 1 1 E 1D 1	get_outer_slug_od() (fuel_buc	* * * * * * * * * * * * * * * * * * * *
get_fresh_u235_mass()	(fuel_bucket.FuelBucket	18 18	ket.Fuelbucket illetilou),
method), 16	/C 1 1 11 E 1D 11	get_port() (module.Module met	thod) 2
get_fresh_U235_mass()	(fuel_bundle.FuelBundle	get_quantity() (phase.Phase me	
method), 23	(first levelest ErralDivalent	get_quantity_history() (phase.P	
get_fresh_u238_mass()	(fuel_bucket.FuelBucket	get_radioactivity() (fuel_bucket	**
method), 16	(fuel_bundle.FuelBundle	get_radioactivity() (fuel_bundle	
get_fresh_u238_mass() method), 23	(luel_bullale.Fuelbullale	get_slug_cladding_volume()	(fuel_bucket.FuelBucket
	at FuelBucket method) 16	method), 19	(1001_00010011 0012 001101
get_fresh_u_mass() (fuel_bucket.FuelBucket method), 16 get_fresh_u_mass() (fuel_bundle.FuelBundle method),		get_slug_fuel_volume()	(fuel_bucket.FuelBucket
24	iic.rucibuildic iiictiiod),	method), 19	( =
	cket FuelBucket method)	get_slug_length() (fuel_bucket.	FuelBucket method), 19
get_fuel_enrichment() (fuel_bucket.FuelBucket method),  17		get_slug_type() (fuel_bucket.Fu	
get_fuel_enrichment() (fuel_bundle.FuelBundle method),		get_solid_phase() (fuel_bundle	
got_raci_cimicimicint() (raci_camarcii acibanarcimicinoa);		get_specie() (fuel_segment.Fue	Segment method), 28

get_species() (fuel_segment.FuelSegment method), 28	Н	
get_value() (phase.Phase method), 37	has_time_stamp() (phase.Phase method), 37	
GetActors() (phase.Phase method), 34	HasGroup() (fuelsegmentsgroups.FuelSegmentsGroup	
GetActors() (stream.Stream method), 49	method), 30	
GetAtoms() (specie.Specie method), 41	heat_pwr (fuel_bucket.FuelBucket attribute), 19	
GetAttribute() (fuelsegments-	heat_pwr (fuel_bundle.FuelBundle attribute), 26	
groups.FuelSegmentsGroups method), 29	···- <u></u>	
GetAttribute() (fuelslug.FuelSlug method), 30		
GetAttribute() (nuclides.Nuclides method), 32	inner_slug_id (fuel_bucket.FuelBucket attribute), 19	
GetCladdingPhase() (fuelslug.FuelSlug method), 31	inner_slug_od (fuel_bucket.FuelBucket attribute), 19	
GetColumn() (phase.Phase method), 34	inner_stug_od (tuet_odeket.1 delbdeket attilodie), 19	
GetFlag() (specie.Specie method), 41	J	
GetFormalName() (quantity.Quantity method), 38		
GetFormula() (specie.Specie method), 41	Jail (class in jail), 8	
GetFormulaName() (specie.Specie method), 41	jail (module), 8	
GetFuelPhase() (fuelslug.FuelSlug method), 31	M	
GetFuelSegments() (fuelsegments-		
groups.FuelSegmentsGroups method), 29	massCC (specie.Specie attribute), 46	
GetMassCC() (specie.Specie method), 41	massCCUnit (specie.Specie attribute), 46	
GetMassCCUnit() (specie.Specie method), 41	Module (class in module), 1	
GetMolarCC() (specie.Specie method), 41	module (module), 1	
GetMolarCCUnit() (specie.Specie method), 41	molarCC (specie.Specie attribute), 46	
GetMolarGammaPwr() (specie.Specie method), 42	molarCCUnit (specie.Specie attribute), 47	
GetMolarGammaPwrUnit() (specie.Specie method), 42	molarGammaPwr (specie.Specie attribute), 47	
GetMolarHeatPwr() (specie.Specie method), 42	molarGammaPwrUnit (specie.Specie attribute), 47	
GetMolarHeatPwrUnit() (specie.Specie method), 42	molarHeatPwr (specie.Specie attribute), 47	
GetMolarMass() (specie.Specie method), 42	molarHeatPwrUnit (specie.Specie attribute), 47	
GetMolarMassUnit() (specie.Specie method), 42	molarMass (specie.Specie attribute), 47	
GetMolarRadioactivity() (specie.Specie method), 42	molarMassUnit (specie.Specie attribute), 47	
GetMolarRadioactivityFractions() (specie.Specie	molarRadioactivity (specie.Specie attribute), 48	
method), 43	molarRadioactivityFractions (specie.Specie attribute), 48	
GetMolarRadioactivityUnit() (specie.Specie method), 43	molarRadioactivityUnit (specie.Specie attribute), 48	
GetName() (specie.Specie method), 43	NI	
GetNAtoms() (specie.Specie method), 43	N	
GetNNuclideTypes() (specie.Specie method), 43	n_fuel_rods (fuel_bundle.FuelBundle attribute), 26	
GetPhase() (specie.Specie method), 43	n_slugs (fuel_bucket.FuelBucket attribute), 19	
GetQuantities() (phase.Phase method), 34	name (fuel_bucket.FuelBucket attribute), 20	
GetQuantities() (stream.Stream method), 49	name (fuel_bundle.FuelBundle attribute), 26	
GetQuantity() (phase.Phase method), 34	name (quantity.Quantity attribute), 39	
GetQuantity() (stream.Stream method), 49	name (specie.Specie attribute), 48	
GetRow() (phase.Phase method), 35	nAtoms (specie.Specie attribute), 48	
GetRow() (stream.Stream method), 49	nNuclideTypes (specie.Specie attribute), 48	
GetSpecie() (phase.Phase method), 35	Nuclides (class in nuclides), 32	
GetSpecie() (stream.Stream method), 49	nuclides (module), 32	
GetSpecies() (phase.Phase method), 35		
GetSpecies() (stream.Stream method), 49	0	
GetSpecs() (fuelslug.FuelSlug method), 31	outer_slug_id (fuel_bucket.FuelBucket attribute), 20	
GetTimeStamp() (stream.Stream method), 50	outer_slug_od (fuel_bucket.FuelBucket attribute), 20	
GetTimeStamps() (phase.Phase method), 35		
GetUnit() (quantity.Quantity method), 38	P	
GetValue() (phase.Phase method), 35	Parole (class in parole), 9	
GetValue() (quantity.Quantity method), 38	parole (module), 9	
GetValue() (stream.Stream method), 50	periodictable (module), 32	
	Phase (class in phase) 33	

phase (module), 33	send() (module.Module method), 2
phase (specie.Specie attribute), 48	send() (port.Port method), 2
plot() (quantity.Quantity method), 40	set_cladding_phase() (fuel_bucket.FuelBucket method),
plot_data (module), 9	20
plot_data() (dataplot.DataPlot method), 6	set_fuel_phase() (fuel_bucket.FuelBucket method), 20
plot_velocity() (vortex.Vortex method), 12	set_fuel_pin_length() (fuel_bundle.FuelBundle method),
PlotData (class in plot_data), 9	26
Port (class in port), 2	set_gas_phase() (fuel_bundle.FuelBundle method), 27
port (module), 2	set_slug_length() (fuel_bucket.FuelBucket method), 20
port_names_expected (module.Module attribute), 1	set_solid_phase() (fuel_bundle.FuelBundle method), 27
Prison (class in prison), 10	set_value() (phase.Phase method), 37
prison (module), 10	SetAtoms() (specie.Specie method), 43
Probation (class in probation), 11	SetFlag() (specie.Specie method), 43
probation (module), 11	SetFormalName() (quantity.Quantity method), 38
Q	SetFormula() (specie.Specie method), 44
	SetFormulaName() (specie.Specie method), 44
quantities (phase.Phase attribute), 37	SetMassCC() (specie.Specie method), 44
Quantity (class in quantity), 38	SetMassCCUnit() (specie.Specie method), 44 SetMolarCC() (specie.Specie method), 44
quantity (module), 38	
D	SetMolarCCUnit() (specie.Specie method), 44 SetMolarGammaPwr() (specie.Specie method), 44
R	SetMolarGammaPwrUnit() (specie.Specie method), 44 SetMolarGammaPwrUnit() (specie.Specie method), 44
radioactivity (fuel_bucket.FuelBucket attribute), 20	SetMolarHeatPwr() (specie.Specie method), 45
radioactivity (fuel_bundle.FuelBundle attribute), 26	SetMolarHeatPwrUnit() (specie.Specie method), 45 SetMolarHeatPwrUnit() (specie.Specie method), 45
recv() (module.Module method), 2	SetMolarMass() (specie.Specie method), 45
recv() (port.Port method), 2	SetMolarMassUnit() (specie.Specie method), 45
recv_data() (dataplot.DataPlot method), 6	SetMolarRadioactivity() (specie.Specie method), 45
ReduceCladdingVolume() (fuelslug.FuelSlug method),	SetMolarRadioactivity() (specie.Specie method), 45 SetMolarRadioactivityFractions() (specie.Specie
31	method), 45
ReduceFuelVolume() (fuelslug.FuelSlug method), 31	SetMolarRadioactivityUnit() (specie.Specie method), 45
RemoveFuelSegment() (fuelsegments-	SetName() (quantity.Quantity method), 38
groups.FuelSegmentsGroups method), 30	SetName() (quantity.Quantity method), 38 SetName() (specie.Specie method), 46
ResetHistory() (phase.Phase method), 35	SetPhase() (specie.Specie method), 46
rhs_fn() (droplet.Droplet method), 6	SetSpecieId() (phase.Phase method), 36
run() (adjudication.Adjudication method), 4	SetSpecieId() (stream.Stream method), 50
run() (arrested.Arrested method), 5	SetUnit() (quantity.Quantity method), 38
run() (community.Community method), 6	SetValue() (phase.Phase method), 36
run() (cortix_main.Cortix method), 1	SetValue() (quantity.Quantity method), 39
run() (dataplot.DataPlot method), 6	Set Value() (stream.Stream method), 50
run() (droplet.Droplet method), 6	slug_cladding_volume (fuel_bucket.FuelBucket at-
run() (dummy_module.DummyModule method), 7	tribute), 21
run() (dummy_module.DummyModule2 method), 8	slug_fuel_volume (fuel_bucket.FuelBucket attribute), 21
run() (jail.Jail method), 8	slug_length (fuel_bucket.FuelBucket attribute), 21
run() (module.Module method), 2	slug_type (fuel_bucket.FuelBucket attribute), 21
run() (parole.Parole method), 9	solid_phase (fuel_bundle.FuelBundle attribute), 27
run() (plot_data.PlotData method), 9	Specie (class in specie), 41
run() (prison.Prison method), 10	specie (fuel_segment.FuelSegment attribute), 28
run() (probation.Probation method), 11	specie (module), 40
run() (vortex.Vortex method), 12	species (fuel_segment.FuelSegment attribute), 28
run_droplet (module), 11	species (phase.Phase attribute), 37
run_justice (module), 11	specs (fuelslug.FuelSlug attribute), 32
S	state (module.Module attribute), 1
	step() (droplet.Droplet method), 7
ScaleRow() (phase.Phase method), 35	Stream (class in stream), 49

```
T
time_stamps (phase.Phase attribute), 37
time_unit (phase.Phase attribute), 37
timeStamps (phase.Phase attribute), 37
timeStamps (phase.Phase attribute), 37

U
unit (quantity.Quantity attribute), 40
V
value (quantity.Quantity attribute), 40
Vortex (class in vortex), 11
vortex (module), 11

W
```

WriteHTML() (phase.Phase method), 36