

---

# RayPyNG

**Simone Vadilonga, Ruslan Ovsyannikov**

**Oct 14, 2022**



**CONTENTS:**

<b>1</b>	<b>Simulate</b>	<b>1</b>
<b>2</b>	<b>SimulationParams</b>	<b>5</b>
<b>3</b>	<b>RayUIRunner</b>	<b>7</b>
<b>4</b>	<b>PostProcessing</b>	<b>9</b>
	<b>Index</b>	<b>13</b>



## SIMULATE

```
class raypyng.simulate.Simulate(rml=None, hide=False, **kwargs)
```

A class that takes care of performing the simulations with RAY-UI

```
__init__(rml=None, hide=False, **kwargs) → None
```

Initialize the class with a rml file :param rml: string pointing to an rml file with the beamline template, or an RMLFile class object. Defaults to None. :type rml: RMLFile/string, optional :param hide: force hiding of GUI leftovers, xvfb needs to be installed. Defaults to False. :type hide: bool, optional

**Raises**

**Exception** – If the rml file is not defined an exception is raised

```
__weakref__
```

list of weak references to the object (if defined)

**property analyze**

Turn on or off the RAY-UI analysis of the results. The analysis of the results takes time, so turn it on only if needed

**Returns**

True: analysis on, False: analysis off

**Return type**

bool

**property exports**

The files to export once the simulation is complete. for a list of possible files check self.possible\_exports and self.possible\_exports\_without\_analysis.

It is expected a list of dictionaries, and for each dictionary the key is the element to be exported and the value are the files to be exported

**property params**

The parameters to scan, as a list of dictionaries. For each dictionary the keys are the parameters elements of the beamline, and the values are the values to be assigned.

**property path**

The path where to execute the simulations

**Returns**

by default the path is the current path from which the program is executed

**Return type**

string

**property possible\_exports**

A list of the files that can be exported by RAY-UI

**Returns**

list of the names of the possible exports for RAY-UI

**Return type**

list

**property possible\_exports\_without\_analysis**

A list of the files that can be exported by RAY-UI when the analysis option is turned off

**Returns**

list of the names of the possible exports for RAY-UI when analysis is off

**Return type**

list

**property raypyng\_analysis**

Turn on or off the RAYPyNG analysis of the results.

**Returns**

True: analysis on, False: analysis off

**Return type**

bool

**property repeat**

The simulations can be repeated an arbitrary number of times If the statistics are not good enough using 2 millions of rays is suggested to repeat them instead of increasing the number of rays

**Returns**

the number of repetition of the simulations, by default is 1

**Return type**

int

**property rml**

RMLFile object instantiated in init

**rml\_list()**

This function creates the folder structure and the rml files to simulate. It requires the param to be set. Useful if one wants to create the simulation files for a manual check before starting the simulations.

**run(recipe=None, /, multiprocessing=True, force=False)**

This method starts the simulations. params and exports need to be defined.

**Parameters**

- **recipe** (*SimulationRecipe*, *optional*) – If using a recipe pass it as a parameter. Defaults to None.
- **multiprocessing** (*boolint*, *optional*) – If True all the cpus are used. If an integer n is provided, n cpus are used. Defaults to True.
- **force** (*bool*, *optional*) – If True all the simulations are performed, even if the export files already exist. If False only the simulations for which are missing some exports are performed. Defaults to False.

**save\_parameters\_to\_file**(*dir*)

save all the user input parameters to file. It takes the values from the SimulationParams class

**Parameters**

**dir** (*str*) – the folder where to save the parameters

**property simulation\_name**

A string to append to the folder where the simulations will be executed.





## SIMULATIONPARAMS

**class** raypyng.simulate.**SimulationParams**(*rml=None, param\_list=None, \*\*kwargs*)

A class that takes care of the simulations parameters, makes sure that they are written correctly, and returns the the list of simulations that is requested by the user.

**\_\_init\_\_**(*rml=None, param\_list=None, \*\*kwargs*) → None

**\_summary\_**

#### Parameters

- **rml** (*RMLFile/string, optional*) – string pointing to an rml file with the beamline template, or an RMLFile class object. Defaults to None.
- **param\_list** (*list, optional*) – list of dictionaries containing the parameters and values to simulate. Defaults to None.

**\_\_weakref\_\_**

list of weak references to the object (if defined)

**\_calc\_loop**(*verbose: bool = True*)

Calculate the simulations loop

#### Returns

independent and dependent parameters self.simulations\_param\_list (list): parameters values for each simulation loop

#### Return type

self.param\_to\_simulate (list)

**\_check\_if\_enabled**(*param*)

Check if a parameter is enabled

#### Parameters

**param** (*RML object*) – an parameter to simulate

#### Returns

True if the parameter is enabled, False otherwise

#### Return type

(bool)

**\_check\_param**()

Check that self.param is a list of dictionaries, and convert the items of the dictionaries to lists, otherwise raise an exception.

**\_enable\_param**(*param*)

Set enabled to True in a beamline object, and auto to False

**Parameters**

**param** (*RML object*) – beamline object

**\_extract\_param**(*verbose: bool = False*)

Parse self.param and extract dependent and independent parameters

**Parameters**

**verbose** (*bool, optional*) – If True print the returned objects. Defaults to False.

**Returns**

indieendent parameter values self.ind\_par (list): independent parameters  
self.dep\_param\_dependency (dict): dictionary of dependencies self.dep\_value\_dependency  
(list): dictionaries of dependent values self.dep\_par (list): dependent parameters

**Return type**

self.ind\_param\_values (list)

**\_write\_value\_to\_param**(*param, value*)

Write a value to a parameter, making sure enable is T and auto is F

**Parameters**

- **param** (*RML object*) – beamline object
- **value** (*str, int, float*) – the value to set the beamline object to

**property params**

The parameters to scan, as a list of dictionaries. For each dictionary the keys are the parameters elements of the beamline, and the values are the values to be assigned.

**property rml**

RMLFile object instantiated in init

## RAYUIRUNNER

**class** raypyng.runner.**RayUIRunner**(*ray\_path=None, ray\_binary='rayui.sh', background=True, hide=False*)

RayUIRunner class implements all logic to start a RayUI process

**\_\_detect\_ray\_path()** → str

Internal function to autodetect installation path of RayUI

**Raises**

**RayPyRunnerError** – is case no ray installations can be detected

**Returns**

string with the detected ray installation path

**Return type**

str

**\_\_init\_\_**(*ray\_path=None, ray\_binary='rayui.sh', background=True, hide=False*) → None

**\_\_weakref\_\_**

list of weak references to the object (if defined)

**\_readline()** → str

read a line from the stdout of the process and convert to a string

**Returns**

line read from the input

**Return type**

str

**\_write**(*instr: str, newline='\n'*)

Write command to RayUI interface

**Parameters**

- **instr** (*str*) – *\_description\_*
- **newline** (*str, optional*) – *\_description\_*. Defaults to newline character.

**Raises**

**RayPyRunnerError** – *\_description\_*

**property isrunning**

Check weather a process is running and rerutn a boolean

**Returns**

returns True if the process is running, otherwise False

**Return type**

bool

**kill()**

kill a RAY-UI process

**property pid**

Get process id of the RayUI process

**Returns**

PID of the process if it running, None otherwise

**Return type**

\_type\_

**run()**

Open one instance of RAY-UI using subprocess

**Raises**

**RayPyRunnerError** – if the RAY-UI executable is not found raise an error

## POSTPROCESSING

**class** raypyng.postprocessing.PostProcess

class to post-process the data. At the moment works only if the exported data are RawRaysOutgoing

**\_\_init\_\_**() → None

**\_\_weakref\_\_**

list of weak references to the object (if defined)

**\_extract\_bandwidth\_fwhm**(rays\_bw: array)

calculate the fwhm of the rays\_bw.

**Parameters**

(**np** (rays\_bw) – array): the energy of the x-rays

**Returns**

fwhm

**Return type**

float

**\_extract\_focus\_fwhm**(rays\_pos: array)

calculate the fwhm of rays\_pos

**Parameters**

**rays\_pos** (**np.array**) – contains positions of the x-rays

**Returns**

fwhm

**Return type**

float

**\_extract\_intensity**(rays: array)

calculate how many rays there are

**Parameters**

**rays** (**np.array**) – contains rays information

**\_list\_files**(dir\_path: str, end\_filename: str)

List all the files in dir\_path ending with end\_filename

**Parameters**

- **dir\_path** (**str**) – path to a folder
- **end\_filename** (**str**) – the listed files end with end\_filename

**Returns**

list of files in `dir_path` ending with `end_filename`

**Return type**

res (list)

**`_load_file(filepath)`**

Load a .npy file and returns the array

**Parameters**

**filepath** (*str*) – the path to the file to load

**Returns**

The loaded numpy array

**Return type**

arr (np.array)

**`_save_file(filename: str, array: array, header: Optional[str] = None)`**

This function is used to save files,

**Parameters**

- **filename** (*str*) – file name(path)
- **array** (*np.array*) – array to save
- **header** (*str*) – header for the file

**`cleanup(dir_path: Optional[str] = None, repeat: int = 1, exp_elements: Optional[list] = None)`**

This function reads all the temporary files created by `self.postprocess_RawRays()` saves one file for each exported element in `dir_path`, and deletes the temporary files. If more than one round of simulations was done, the values are averaged.

**Parameters**

- **dir\_path** (*str*, *optional*) – The path to the folder to cleanup. Defaults to `None`.
- **repeat** (*int*, *optional*) – number of rounds of simulations. Defaults to 1.
- **exp\_elements** (*list*, *optional*) – the exported elements names as *str*. Defaults to `None`.

**`extract_nrays_from_source(rml_filename)`**

Extract photon flux from rml file, find source automatically

**Parameters**

**rml\_filename** (*str*) – the rml file to use to extract the photon flux

**Returns**

the photon flux

**Return type**

str

**`postprocess_RawRays(exported_element: Optional[str] = None, exported_object: Optional[str] = None, dir_path: Optional[str] = None, sim_number: Optional[str] = None, rml_filename: Optional[str] = None)`**

The method looks in the folder `dir_path` for a file with the filename: `filename = os.path.join(dir_path, sim_number+exported_element + '-' + exported_object+'.csv')` for each file it calculates the number of rays, the bandwidth, and the horizontal and vertical focus size, it saves it in an array that is composed by `[n_rays, bandwidth, hor_focus, vert_focus]`, that is then saved to `os.path.join(dir_path, sim_number+exported_element+'_analyzed_rays.npy')` :param `exported_element`:

a list of containing the exported elements name as str. Defaults to None. :type exported\_element: list, optional :param exported\_object: the exported object, tested only with Raw Rays Outgoing. Defaults to None. :type exported\_object: str, optional :param dir\_path: the folder where the file to process is located. Defaults to None. :type dir\_path: str, optional :param sim\_number: the prefix of the file, that is the simulation number with a \_prepended, ie “**0**\_  ”. Defaults to None. :type sim\_number: str, optional





## Symbols

- `__detect_ray_path()` (*raypyng.runner.RayUIRunner method*), 7
  - `__init__()` (*raypyng.postprocessing.PostProcess method*), 9
  - `__init__()` (*raypyng.runner.RayUIRunner method*), 7
  - `__init__()` (*raypyng.simulate.Simulate method*), 1
  - `__init__()` (*raypyng.simulate.SimulationParams method*), 5
  - `__weakref__` (*raypyng.postprocessing.PostProcess attribute*), 9
  - `__weakref__` (*raypyng.runner.RayUIRunner attribute*), 7
  - `__weakref__` (*raypyng.simulate.Simulate attribute*), 1
  - `__weakref__` (*raypyng.simulate.SimulationParams attribute*), 5
  - `_calc_loop()` (*raypyng.simulate.SimulationParams method*), 5
  - `_check_if_enabled()` (*raypyng.simulate.SimulationParams method*), 5
  - `_check_param()` (*raypyng.simulate.SimulationParams method*), 5
  - `_enable_param()` (*raypyng.simulate.SimulationParams method*), 5
  - `_extract_bandwidth_fwhm()` (*raypyng.postprocessing.PostProcess method*), 9
  - `_extract_focus_fwhm()` (*raypyng.postprocessing.PostProcess method*), 9
  - `_extract_intensity()` (*raypyng.postprocessing.PostProcess method*), 9
  - `_extract_param()` (*raypyng.simulate.SimulationParams method*), 6
  - `_list_files()` (*raypyng.postprocessing.PostProcess method*), 9
  - `_load_file()` (*raypyng.postprocessing.PostProcess method*), 10
  - `_readline()` (*raypyng.runner.RayUIRunner method*), 7
  - `_save_file()` (*raypyng.postprocessing.PostProcess method*), 10
  - `_write()` (*raypyng.runner.RayUIRunner method*), 7
  - `_write_value_to_param()` (*raypyng.simulate.SimulationParams method*), 6
- ## A
- `analyze` (*raypyng.simulate.Simulate property*), 1
- ## C
- `cleanup()` (*raypyng.postprocessing.PostProcess method*), 10
- ## E
- `exports` (*raypyng.simulate.Simulate property*), 1
  - `extract_nrays_from_source()` (*raypyng.postprocessing.PostProcess method*), 10
- ## I
- `isrunning` (*raypyng.runner.RayUIRunner property*), 7
- ## K
- `kill()` (*raypyng.runner.RayUIRunner method*), 8
- ## P
- `params` (*raypyng.simulate.Simulate property*), 1
  - `params` (*raypyng.simulate.SimulationParams property*), 6
  - `path` (*raypyng.simulate.Simulate property*), 1
  - `pid` (*raypyng.runner.RayUIRunner property*), 8
  - `possible_exports` (*raypyng.simulate.Simulate property*), 1
  - `possible_exports_without_analysis` (*raypyng.simulate.Simulate property*), 2
  - `PostProcess` (*class in raypyng.postprocessing*), 9
  - `postprocess_RawRays()` (*raypyng.postprocessing.PostProcess method*), 10
- ## R
- `raypyng_analysis` (*raypyng.simulate.Simulate property*), 2

`RayUIRunner` (*class in raypyng.runner*), 7  
`repeat` (*raypyng.simulate.Simulate property*), 2  
`rml` (*raypyng.simulate.Simulate property*), 2  
`rml` (*raypyng.simulate.SimulationParams property*), 6  
`rml_list()` (*raypyng.simulate.Simulate method*), 2  
`run()` (*raypyng.runner.RayUIRunner method*), 8  
`run()` (*raypyng.simulate.Simulate method*), 2

## S

`save_parameters_to_file()`  
    (*raypyng.simulate.Simulate method*), 2  
`Simulate` (*class in raypyng.simulate*), 1  
`simulation_name` (*raypyng.simulate.Simulate property*), 3  
`SimulationParams` (*class in raypyng.simulate*), 5