RayPyNG

Simone Vadilonga, Ruslan Ovsyannikov

Aug 12, 2022

CONTENTS:

CHAPTER

ONE

SIMULATE

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

class to simulate

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

summary

Parameters

- **rml** (*_type_, optional*) – Rml file with the beamline template. Defaults to None.
- **hide** (*bool, optional*) – force hiding of GUI leftovers. Defaults to False.

Raises

Exception – _description_

__weakref__

list of weak references to the object (if defined)

CHAPTER

TWO

SIMULATIONPARAMS

class raypyng.simulate.**SimulationParams**(*rml=None*, *param_list=None*, ***kwargs*)

__init__(*rml=None*, *param_list=None*, ***kwargs*) → None

__weakref__

list of weak references to the object (if defined)

_calc_loop(*verbose: bool = True*)

Calculate the simulations loop

Returns
idependent 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

CHAPTER

THREE

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*, *endline='\n'*)

Write command to RayUI interface

Parameters

- **instr** (*str*) – _description_
- **endline** (*str, optional*) – _description_. Defaults to endline 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

INDEX

Symbols

I

K

P

R

S