RayPyNG

Simone Vadilonga, Ruslan Ovsyannikov

Sep 23, 2022

CONTENTS:

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: Rml file with the beamline template. Defaults to None. :type rml: 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 expeceted a list of dictionaries, and for each dictionary the key is the element to be exported and the valuee 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 simlations

        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 repeat

The simulations can be repeated an arbitrary number of times If the statitcs 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

Returns

RMLFile

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 recipee 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 simlations are performed, even if the export files already exist. If False only the simlations for which are missing some exports are performed. Defaults to False.

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*)

    __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

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