raypyng-bluesky

Simone Vadilonga, Ruslan Ovsyannikov

Nov 28, 2022

CONTENTS:

CHAPTER

ONE

INSTALLATION

CHAPTER TWO

TUTORIAL

CHAPTER

THREE

HOW TO GUIDES

CHAPTER

FOUR

API

4.1 Create Ophyd Devices from rml file

4.1.1 RaypyngOphydDevices

class raypyng_bluesky.RaypyngOphydDevices.**RaypyngOphydDevices**(**args*, *RE*, *rml_path*, *temporary_folder=None*, *name_space=None*, *prefix=None*, ***kwargs*)

Create ophyd devices from a RAY-UI rml file and adds them to a name space.

If you are using ipython sys._getframe(0) returns the name space of the ipython instance. (Remember to `import sys`)

Parameters

- **RE** (*RunEngine*) – Bluesky RunEngine
- **rml_path** (*str*) – the path to the rml file
- **temporary_folder** (*str*) – path where to create temporary folder. If None it is automatically set into the ipython profile folder. Default to None.
- **name_space** (*frame, optional*) – If None the class will try to understand the correct namespace to add the Ophyd devices to. If the automatic retrieval fails, pass``sys._getframe(0)``. Defaults to None.
- **prefix** (*str*) – the prefix to prepend to the oe names found in the rml file

append_preprocessor()

Add supplemental data to the RunEngine to trigger the simulations

create_raypyng_elements_from_rml()

Iterate through the raypyng objects created by RMLFile and create corresponding Ophyd Devices

Returns
the Ophyd devices created

Return type
OphydDevices

create_trigger_detector()

Create a trigger detector called RaypyngTriggerDetector

prepend_to_oe_name()

Prepend a prefix to the name of all the Ophyd object created

trigger_detector()

Return the RaypyngTriggerDetector

Returns

the trigger detector

Return type

RaypyngTriggerDetector (*RaypyngTriggerDetector*)

4.1.2 RaypyngDictionary

`class raypyng_bluesky.RaypyngOphydDevices.`**`RaypyngDictionary`**`(`**args*, ***kwargs*`)`

A class defining a dictionary of the differen elements in rayui and the classe to be used as Ophyd devices

4.2 Ophyd Signals

4.2.1 RayPySignal

`class raypyng_bluesky.signal.`**`RayPySignal`**`(`**args*, ***kwargs*`)`

get()

The readback value

put(**args*, ***kwargs*)

Put updates the internal readback value

The value is optionally checked first, depending on the value of force. In addition, VALUE subscriptions are run.

Extra kwargs are ignored (for API compatibility with EpicsSignal kwargs pass through).

Parameters

- **value** (*any*) – Value to set
- **timestamp** (*float, optional*) – The timestamp associated with the value, defaults to time.time()
- **metadata** (*dict, optional*) – Further associated metadata with the value (such as alarm status, severity, etc.)
- **force** (*bool, optional*) – Check the value prior to setting it, defaults to False

set(*value*)

Set is like *put*, but is here for bluesky compatibility

Returns

st – This status object will be finished upon return in the case of basic soft Signals

Return type

Status

4.2.2 RayPySignalRO

`class raypyng_bluesky.signal.`**`RayPySignalRO`**`(`**args*, ***kwargs*`)`

`put``(`*value*, ***, *timestamp=None*, *force=False*`)`

Put updates the internal readback value

The value is optionally checked first, depending on the value of force. In addition, VALUE subscriptions are run.

Extra kwargs are ignored (for API compatibility with EpicsSignal kwargs pass through).

Parameters

- **`value`** (*`any`*) – Value to set
- **`timestamp`** (*`float, optional`*) – The timestamp associated with the value, defaults to time.time()
- **`metadata`** (*`dict, optional`*) – Further associated metadata with the value (such as alarm status, severity, etc.)
- **`force`** (*`bool, optional`*) – Check the value prior to setting it, defaults to False

`set``(`*value*, ***, *timestamp=None*, *force=False*`)`

Set is like *put*, but is here for bluesky compatibility

Returns

st – This status object will be finished upon return in the case of basic soft Signals

Return type

Status

4.3 Ophyd Devices

4.3.1 Axes

`class raypyng_bluesky.axes.`**`RaypyngAxis`**`(`**args*, ***kwargs*`)`

`get``()`

Get the value of all components in the device

Keyword arguments are passed onto each signal.get(). Components beginning with an underscore will not be included.

`property position`

The current position of the motor in its engineering units :returns: **position** :rtype: any

`set``(`*value*`)`

Set a value and return a Status object

Parameters

- **`new_position`** (*`object`*) – The input here is whatever the device requires (this should be over-ridden by the implementation. For example a motor would take a float, a shutter the strings {'Open', 'Close'}, and a goineometer (h, k, l) tuples
- **`timeout`** (*`float, optional`*) – Maximum time to wait for the motion. If None, the default timeout for this positioner is used.

- **moved_cb** (*callable, optional*) – Deprecated

  Call this callback when movement has finished. This callback must accept one keyword argument: 'obj' which will be set to this positioner instance.

- **wait** (*bool, optional*) – Deprecated

  If the method should block until the Status object reports it is done.

  Defaults to False

Returns
status – Status object to indicate when the motion / set is done.

Return type
StatusBase

4.3.2 SimulatedAxisSource

`class raypyng_bluesky.axes.SimulatedAxisSource(*args, **kwargs)`

class SimulatedAxisMisalign(RaypyngAxis):

4.3.3 SimulatedAxisAperture

`class raypyng_bluesky.axes.SimulatedAxisAperture(*args, **kwargs)`

4.3.4 SimulatedAxisGrating

`class raypyng_bluesky.axes.SimulatedAxisGrating(*args, **kwargs)`

4.4 Ophyd Detectors

4.4.1 Detector

`class raypyng_bluesky.detector.RaypyngDetector(*args, information_to_extract='intensity', parent_detector_name=None, **kwargs)`

`get()`

The readback value

`put(value, *, timestamp=None, force=False)`

Put updates the internal readback value

The value is optionally checked first, depending on the value of force. In addition, VALUE subscriptions are run.

Extra kwargs are ignored (for API compatibility with EpicsSignal kwargs pass through).

Parameters

- **value** (*any*) – Value to set

- **timestamp** (*float, optional*) – The timestamp associated with the value, defaults to time.time()
- **metadata** (*dict, optional*) – Further associated metadata with the value (such as alarm status, severity, etc.)
- **force** (*bool, optional*) – Check the value prior to setting it, defaults to False

set(*value*, ***, *timestamp=None*, *force=False*)

Set is like *put*, but is here for bluesky compatibility

Returns
st – This status object will be finished upon return in the case of basic soft Signals

Return type
Status

trigger()

Call that is used by bluesky prior to read()

4.4.2 Trigger Detector

class raypyng_bluesky.detector.**RaypyngTriggerDetector**(**args*, *rml*, *temporary_folder*, ***kwargs*)

get()

The readback value

put(*value*, ***, *timestamp=None*, *force=False*)

Put updates the internal readback value

The value is optionally checked first, depending on the value of force. In addition, VALUE subscriptions are run.

Extra kwargs are ignored (for API compatibility with EpicsSignal kwargs pass through).

Parameters

- **value** (*any*) – Value to set
- **timestamp** (*float, optional*) – The timestamp associated with the value, defaults to time.time()
- **metadata** (*dict, optional*) – Further associated metadata with the value (such as alarm status, severity, etc.)
- **force** (*bool, optional*) – Check the value prior to setting it, defaults to False

set(*value*, ***, *timestamp=None*, *force=False*)

Set is like *put*, but is here for bluesky compatibility

Returns
st – This status object will be finished upon return in the case of basic soft Signals

Return type
Status

trigger()

Call that is used by bluesky prior to read()

4.5 Ophyd Devices

4.5.1 MisalignComponents

`class raypyng_bluesky.devices.`**`MisalignComponents`**`(`*`*args`*`, `*`obj`*`, `*`**kwargs`*`)`

4.5.2 SimulatedPGM

`class raypyng_bluesky.devices.`**`SimulatedPGM`**`(`*`*args`*`, `*`obj`*`, `*`**kwargs`*`)`

4.5.3 SimulatedApertures

`class raypyng_bluesky.devices.`**`SimulatedApertures`**`(`*`*args`*`, `*`obj`*`, `*`**kwargs`*`)`

4.5.4 SimulatedMirror

`class raypyng_bluesky.devices.`**`SimulatedMirror`**`(`*`*args`*`, `*`obj`*`, `*`**kwargs`*`)`

4.5.5 SimulatedSource

`class raypyng_bluesky.devices.`**`SimulatedSource`**`(`*`*args`*`, `*`obj`*`, `*`**kwargs`*`)`

4.6 Preprocessor

4.6.1 MisalignComponents

`raypyng_bluesky.preprocessor.`**`trigger_sim`**`(`*`plan`*`, `*`trigger_detector`*`)`

Trigger simulations for raypyng plans

This function is composed of four steps: 1- populate_raypyng_devices_list_at_stage:

> at the 'stage message' each device is classified and saved into two list. One list is dedicated to raypng devices, and one for all the others

2- prepare_simulations_at_open_run:
when the message is 'open_run', if both raypyng devices and normal devices have been staged raise an exception. Otherwise the list of exports is prepared(consists of detector names included in the plan) and passed to the trigger detector. The done simulation file is removed from the temporary folder.

3- insert_before_first_det_trigger:
before the first detector is triggered, a trigger message for the raypyng trigger detector is inserted in the same group as the other detectors

4- cleanup_at_close_run:
when the message is 'close_run' the simulation_done file is removed and the list containing the raypyng and other devices, created at point 1, are cleared.

Parameters

- **plan** (*bluesky.plan*) – the plan that is being executed
- **trigger_detector** (RaypyngTriggerDetector) – the trigger detector

Raises

- **ValueError** – if in the plan a mix of raypyng devices are other devices
- **are used raise an exeption** –

4.6.2 SupplementalDataRaypyng

class raypyng_bluesky.preprocessor.**SupplementalDataRaypyng**(**args*, *trigger_detector*, ***kwargs*)

Supplemental data for raypyng.

The Run engine is needed to be able to include the trigger detector automatically

Parameters

trigger_detector (RaypyngTriggerDetector) – The detector to trigger raypyng

INDEX