# **4.2.9. Python**

The Python component allows interaction with Python 3 scripts.

## **Properties**

|  |  |
| --- | --- |
| Property | Description |
| **Model coupling settings** | **General Co-Simulation Element Settings** |
| **Model settings** |  |
| File | Path to the Python file (PY) that defines the interface and contains simulation code.  ***Restriction:*** Python files referenced in Python elements inside the same model must have unique names. The file name must start with either a letter or an underscore, and be composed entirely of letters, digits and/or underscores. |
| Python library | Path to the Python library (DLL, SO) used in simulation. By default, the Python 3 library from the installation is used. |
| Python executable | Path to the Python executable (EXE) used to gather interface information. By default, the Python 3 executable from the installation is used. If a Python library is set and an executable can easily be found the the same directory, this will become the new default. |
| Python home | Path to the Python home directory used in simulation. By default, the Python 3 home directory from the installation is used. If a Python library is set, its parent directory will become the new default. |
| **Execution group** | **General Co-Simulation Element Settings** |

## **Interface Definition**

The selected Python file needs to contain a class named PythonInterface. This class will be instantiated and the following methods called:

* configure()
* initialize()
* do\_step(self, current\_communication\_point, communication\_step\_size):

\_\_init\_\_ and configure will be called when the file is loaded, initialize at the start of each simulation and do\_step for each simulation step.

Interface class template:

class PythonInterface:

def \_\_init\_\_(self):

# TODO implement object initialization

pass

def configure(self):

# TODO implement object configuration

pass

def initialize(self):

# TODO implement simulation initialization

pass

def do\_step(self, current\_communication\_point, communication\_step\_size):

# TODO implement simulation step

pass

## **Ports**

Input and output ports of an element need to be defined as inputs and outputs attributes of an interface object. They both have to be dict objects with port names as keys and port definitions as values.

Port names have to be unique non-empty strings. Port definitions have to be dict objects with string keys and values of primitive types (float, int, str, bool or None).

The following port definition entries will be used:

* type
* value
* unit

The definition of type is mandatory and has to be one of the following strings:

* float
* int
* bool
* string
* binary

If defined, the type of value has to correspond to the port type. Float ports support float and int values, integers ports support int values, boolean ports support bool values, and string ports support str values. Binary ports will not have their values shown in the UI, but they will be used in simulation. The values will be set to the Default value field of input ports and the Initial value field of output ports.

Furthermore, unit will only be used for float ports. It needs to be a string containing an SDT unit ID (see **List of SDT Unit IDs**). If not defined, no\_unit~none will be used.

Example:

def configure(self):

self.inputs = {'amount\_in': {'type': 'float',

'value': 5.7,

'unit': 'length~m'},

'float\_in': {'type': 'float',

'value': 2.6},

'int\_in': {'type': 'int',

'value': 7}}

self.outputs = {'bool\_out': {'type': 'bool',

'value': True},

'string\_out': {'type': 'string',

'value': 'text'},

'binary\_out': {'type': 'binary'}}

## **Parameters**

Parameters of an element need to be defined as the parameters attribute of an interface object. It has to be a dict object with parameter names as keys and parameter definitions as values.

Parameter names have to be unique non-empty strings. Parameter definitions have to be dict objects with string keys and values of primitive types (float, int, str, bool or None).

The following parameter definition entries will be used:

* type
* value
* unit

The definition of type is mandatory and has to be one of the following strings:

* float
* int
* bool
* string

The definition of value is also mandatory and has to correspond to the parameter type. Float parameters support float and int values, integers parameters support int values, boolean parameters support bool values, and string parameters support str values.

Example:

def configure(self):

self.parameters = {'amount': {'type': 'float',

'value': 5.7,

'unit': 'length~m'},

'float': {'type': 'float',

'value': 2.6},

'int': {'type': 'int',

'value': 7},

'bool': {'type': 'bool',

'value': True},

'string': {'type': 'string',

'value': 'text'}}