

---

# **agentMET4FOF Documentation**

**Bang Xiang Yong**

**Jul 03, 2020**



---

## Getting started:

---

<b>1</b>	<b>Multi-Agent System for Metrology for Factory of the Future (Met4FoF) Code</b>	<b>3</b>
<b>2</b>	<b>Tutorial 1 - A simple pipeline to plot a signal</b>	<b>9</b>
<b>3</b>	<b>Tutorial 2 - A simple pipeline with signal postprocessing.</b>	<b>13</b>
<b>4</b>	<b>Tutorial 3 - An advanced pipeline with multichannel signals.</b>	<b>17</b>
<b>5</b>	<b>agentMET4FOF agents</b>	<b>21</b>
<b>6</b>	<b>agentMET4FOF streams</b>	<b>29</b>
<b>7</b>	<b>agentMET4FOF dashboard</b>	<b>31</b>
<b>8</b>	<b>Indices and tables</b>	<b>33</b>
<b>9</b>	<b>References</b>	<b>35</b>
	<b>Bibliography</b>	<b>37</b>
	<b>Python Module Index</b>	<b>39</b>
	<b>Index</b>	<b>41</b>



*agentMET4FOF* is a Python library developed at the [Institute for Manufacturing](#) of the University of Cambridge (UK) as part of the European joint Research Project [EMPIR 17IND12 Met4FoF](#).

For the *agentMET4FOF* homepage go to [GitHub](#).

*agentMET4FOF* is written in Python 3.

[CircleCI](#) [Documentation](#) [Status](#) [Codecov](#) [Badge](#)



---

## Multi-Agent System for Metrology for Factory of the Future (Met4FoF) Code

---

This is supported by European Metrology Programme for Innovation and Research (EMPIR) under the project Metrology for the Factory of the Future (Met4FoF), project number 17IND12. (<https://www.ptb.de/empir2018/met4fof/home/>)

### 1.1 About

- How can metrological input be incorporated into an agent-based system for addressing uncertainty of machine learning in future manufacturing?
- Includes agent-based simulation and implementation
- Readthedocs documentation is available at (<https://agentmet4fof.readthedocs.io>)

### 1.2 Use agentMET4FOF

The easiest way to get started with *agentMET4FOF* is navigating to the folder in which you want to create a virtual Python environment (*venv*), create one based on Python 3.6 or later, activate it, first install *numpy*, then install *agentMET4FOF* from PyPI.org and then work through the [tutorials](#) or [examples](#). To do this, issue the following commands on your Shell:

```
$ cd /LOCAL/PATH/TO/ENVS
$ python3 -m venv agentMET4FOF_venv
$ source agentMET4FOF_venv/bin/activate
(agentMET4FOF_venv) $ pip install agentMET4FOF
Collecting agentMET4FOF
...
Successfully installed agentMET4FOF-... ...
(agentMET4FOF_venv) $ python
```

(continues on next page)

(continued from previous page)

```

Python ... (default, ..., ...)
[GCC ...] on ...
Type "help", "copyright", "credits" or "license" for more information.
>>> from agentMET4FOF_tutorials import tutorial_1_generator_agent
>>> tutorial_1_generator_agent.demonstrate_generator_agent_use()
Starting NameServer...
Broadcast server running on 0.0.0.0:9091
NS running on 127.0.0.1:3333 (127.0.0.1)
URI = PYRO:Pyro.NameServer@127.0.0.1:3333
INFO [2020-02-21 19:04:26.961014] (AgentController): INITIALIZED
INFO [2020-02-21 19:04:27.032258] (Logger): INITIALIZED
* Serving Flask app "agentMET4FOF.dashboard.Dashboard" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
...

```

Now you can visit <http://127.0.0.1:8050/> with any Browser and watch the SineGenerator agent you just spawned.

To get some insights and really get going please visit [agentMET4FOF.readthedocs.io](http://agentMET4FOF.readthedocs.io).

## 1.3 Get started developing

First clone the repository to your local machine as described [here](#). To get started with your present *Anaconda* installation just go to *Anaconda prompt*, navigate to your local clone

```
cd /LOCAL/PATH/TO/agentMET4FOF
```

and execute

```
conda env create --file environment.yml
```

This will create an *Anaconda* virtual environment with all dependencies satisfied. If you don't have *Anaconda* installed already follow [this guide](#) first, then create the virtual environment as stated above and then proceed.

Alternatively, for non-conda environments, you can install the dependencies using pip

```
pip install -r requirements.txt
```

First take a look at the [tutorials](#) and [examples](#) or start hacking if you already are familiar with agentMET4FOF and want to customize your agents' network.

Alternatively, watch the tutorial webinar [here](#)

## 1.4 Updates

- Implemented base class AgentMET4FOF with built-in agent classes DataStreamAgent, MonitorAgent
- Implemented class AgentNetwork to start or connect to a agent server
- Implemented with ZEMA prognosis of Electromechanical cylinder data set as use case [DOI](#)



- Implemented interactive web application with user interface



1.5 Screenshot of web visualization



## 1.6 Note

- In the event of agents not terminating cleanly, run

```
taskkill /f /im python.exe /t
```

in Windows Command Prompt to terminate all background python processes.

---

### Tutorial 1 - A simple pipeline to plot a signal

---

First we define a simple pipeline of two agents, of which one will generate a signal (in our case a *SineGeneratorAgent*) and the other one plots the signal on the dashboard (this is always a *MonitorAgent*).

We define a *SineGeneratorAgent* for which we have to override the functions `init_parameters()` & `agent_loop()` to define the new agent's behaviour.

- `init_parameters()` is used to setup the input data stream and potentially other necessary parameters.
- `agent_loop()` will be endlessly repeated until further notice. It will sample by sample extract the input data stream's content and push it to all agents connected to *SineGeneratorAgent*'s output channel by invoking `send_output()`.

The *MonitorAgent* is connected to the *SineGeneratorAgent*'s output channel and per default automatically plots the output.

Each agent has an internal `current_state` which can be used as a switch to change the behaviour of the agent. The available states are listed [here](#).

As soon as all agents are initialized and the connections are set up, the agent network is started by accordingly changing all agents' state simultaneously.

```
[1]: # %load tutorial_1_generator_agent.py
from agentMET4FOF.agents import AgentMET4FOF, AgentNetwork, MonitorAgent
from agentMET4FOF.streams import SineGenerator

class SineGeneratorAgent(AgentMET4FOF):
    """An agent streaming a sine signal

    Takes samples from the :py:mod:`SineGenerator` and pushes them sample by sample
    to connected agents via its output channel.
    """
    _sine_stream: SineGenerator

    def init_parameters(self):
        """Initialize the input data
```

(continues on next page)

(continued from previous page)

```

        Initialize the input data stream as an instance of the
        :py:mod:`SineGenerator` class
        """
        self._sine_stream = SineGenerator()

    def agent_loop(self):
        """Model the agent's behaviour

        On state *Running* the agent will extract sample by sample the input data
        streams content and push it via invoking :py:method:`AgentMET4FOF.send_output`.
        """
        if self.current_state == "Running":
            sine_data = self._sine_stream.next_sample() # dictionary
            self.send_output(sine_data["x"])

def demonstrate_generator_agent_use():
    # Start agent network server.
    agent_network = AgentNetwork()

    # Initialize agents by adding them to the agent network.
    gen_agent = agent_network.add_agent(agentType=SineGeneratorAgent)
    monitor_agent = agent_network.add_agent(agentType=MonitorAgent)

    # Interconnect agents by either way:
    # 1) by agent network.bind_agents(source, target).
    agent_network.bind_agents(gen_agent, monitor_agent)

    # 2) by the agent.bind_output().
    gen_agent.bind_output(monitor_agent)

    # Set all agents' states to "Running".
    agent_network.set_running_state()

    # Allow for shutting down the network after execution
    return agent_network

if __name__ == "__main__":
    demonstrate_generator_agent_use()

```

```

Starting NameServer...
Broadcast server running on 0.0.0.0:9091
NS running on 127.0.0.1:3333 (127.0.0.1)
URI = PYRO:Pyro.NameServer@127.0.0.1:3333
INFO [2020-04-24 09:21:23.002156] (AgentController): INITIALIZED
INFO [2020-04-24 09:21:23.200110] (SineGeneratorAgent_1): INITIALIZED
* Serving Flask app "agentMET4FOF.dashboard.Dashboard" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
INFO [2020-04-24 09:21:23.294149] (MonitorAgent_1): INITIALIZED
[2020-04-24 09:21:23.360214] (SineGeneratorAgent_1): Connected output module:
↳MonitorAgent_1

```

(continues on next page)

(continued from previous page)

```

SET STATE:    Running
[2020-04-24 09:21:24.218709] (SineGeneratorAgent_1): Pack time: 0.000897
[2020-04-24 09:21:24.223207] (SineGeneratorAgent_1): Sending: [0.]
[2020-04-24 09:21:24.225895] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
→1', 'data': array([0.]), 'senderType': 'SineGeneratorAgent', 'channel': 'default'}
[2020-04-24 09:21:24.226708] (MonitorAgent_1): Tproc: 0.00017
[2020-04-24 09:21:25.217698] (SineGeneratorAgent_1): Pack time: 0.000419
[2020-04-24 09:21:25.220045] (SineGeneratorAgent_1): Sending: [0.47942554]
[2020-04-24 09:21:25.221406] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
→1', 'data': array([0.47942554]), 'senderType': 'SineGeneratorAgent', 'channel':
→'default'}
[2020-04-24 09:21:25.223795] (MonitorAgent_1): Memory: {'SineGeneratorAgent_1':_
→array([0.          , 0.47942554])}
[2020-04-24 09:21:25.224696] (MonitorAgent_1): Tproc: 0.00264
[2020-04-24 09:21:26.217785] (SineGeneratorAgent_1): Pack time: 0.000415
[2020-04-24 09:21:26.218995] (SineGeneratorAgent_1): Sending: [0.84147098]
[2020-04-24 09:21:26.220107] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
→1', 'data': array([0.84147098]), 'senderType': 'SineGeneratorAgent', 'channel':
→'default'}
[2020-04-24 09:21:26.222038] (MonitorAgent_1): Memory: {'SineGeneratorAgent_1':_
→array([0.          , 0.47942554, 0.84147098])}
[2020-04-24 09:21:26.223223] (MonitorAgent_1): Tproc: 0.002481
[2020-04-24 09:21:27.216058] (SineGeneratorAgent_1): Pack time: 0.000131
[2020-04-24 09:21:27.216323] (SineGeneratorAgent_1): Sending: [0.99749499]
[2020-04-24 09:21:27.216876] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
→1', 'data': array([0.99749499]), 'senderType': 'SineGeneratorAgent', 'channel':
→'default'}
[2020-04-24 09:21:27.217220] (MonitorAgent_1): Memory: {'SineGeneratorAgent_1':_
→array([0.          , 0.47942554, 0.84147098, 0.99749499])}
[2020-04-24 09:21:27.217288] (MonitorAgent_1): Tproc: 0.000314
[2020-04-24 09:21:28.215905] (SineGeneratorAgent_1): Pack time: 0.000102
[2020-04-24 09:21:28.216367] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
→1', 'data': array([0.90929743]), 'senderType': 'SineGeneratorAgent', 'channel':
→'default'}
[2020-04-24 09:21:28.216186] (SineGeneratorAgent_1): Sending: [0.90929743]
[2020-04-24 09:21:28.216623] (MonitorAgent_1): Memory: {'SineGeneratorAgent_1':_
→array([0.          , 0.47942554, 0.84147098, 0.99749499, 0.90929743])}
[2020-04-24 09:21:28.216678] (MonitorAgent_1): Tproc: 0.000229

```

\* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)

<Figure size 432x288 with 0 Axes>





---

## Tutorial 2 - A simple pipeline with signal postprocessing.

---

Here we demonstrate how to build a *MathAgent* as an intermediate to process the *SineGeneratorAgent*'s output before plotting. Subsequently, a *MultiMathAgent* is built to show the ability to send a dictionary of multiple fields to the recipient. We provide a custom `on_received_message()` function, which is called every time a message is received from input agents.

The received message is a dictionary of the form:

```
{
  'from': agent_name,
  'data': data,
  'senderType': agent_class_name,
  'channel': 'channel_name'
}
```

By default, 'channel' is set to "default", however a custom channel can be set when needed, which is demonstrated in the next tutorial.

```
[ ]: # %load tutorial_2_math_agent.py
from agentMET4FOF.agents import AgentMET4FOF, AgentNetwork, MonitorAgent
from agentMET4FOF.streams import SineGenerator

# Define simple math functions.
def divide_by_two(numerator: float) -> float:
    return numerator / 2

def minus(minuend: float, subtrahend: float) -> float:
    return minuend - subtrahend

def plus(summand_1: float, summand_2: float) -> float:
    return summand_1+summand_2
```

(continues on next page)

(continued from previous page)

```

class MathAgent (AgentMET4FOF) :
    def on_received_message (self, message) :
        data = divide_by_two (message['data'])
        self.send_output (data)

class MultiMathAgent (AgentMET4FOF) :
    def init_parameters (self, minus_param=0.5, plus_param=0.5) :
        self.minus_param = minus_param
        self.plus_param = plus_param

    def on_received_message (self, message) :
        minus_data = minus (message['data'], self.minus_param)
        plus_data = plus (message['data'], self.plus_param)

        self.send_output ({'minus':minus_data, 'plus':plus_data})

class SineGeneratorAgent (AgentMET4FOF) :
    def init_parameters (self) :
        self.stream = SineGenerator()

    def agent_loop (self) :
        if self.current_state == "Running":
            sine_data = self.stream.next_sample() #dictionary
            self.send_output (sine_data['x'])

def main() :
    # start agent network server
    agentNetwork = AgentNetwork()
    # init agents
    gen_agent = agentNetwork.add_agent (agentType=SineGeneratorAgent)
    math_agent = agentNetwork.add_agent (agentType=MathAgent)
    multi_math_agent = agentNetwork.add_agent (agentType=MultiMathAgent)
    monitor_agent = agentNetwork.add_agent (agentType=MonitorAgent)
    # connect agents : We can connect multiple agents to any particular agent
    agentNetwork.bind_agents (gen_agent, math_agent)
    agentNetwork.bind_agents (gen_agent, multi_math_agent)
    # connect
    agentNetwork.bind_agents (gen_agent, monitor_agent)
    agentNetwork.bind_agents (math_agent, monitor_agent)
    agentNetwork.bind_agents (multi_math_agent, monitor_agent)
    # set all agents states to "Running"
    agentNetwork.set_running_state()

    # allow for shutting down the network after execution
    return agentNetwork

if __name__ == '__main__':
    main()

```

```

Starting NameServer...
Broadcast server running on 0.0.0.0:9091

```

(continues on next page)

(continued from previous page)

```
NS running on 127.0.0.1:3333 (127.0.0.1)
URI = PYRO:Pyro.NameServer@127.0.0.1:3333
INFO [2020-07-03 10:31:04.994115] (AgentController): INITIALIZED
INFO [2020-07-03 10:31:05.139466] (SineGeneratorAgent_1): INITIALIZED
* Serving Flask app "agentMET4FOF.dashboard.Dashboard" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
INFO [2020-07-03 10:31:05.228778] (MathAgent_1): INITIALIZED
INFO [2020-07-03 10:31:05.331405] (MultiMathAgent_1): INITIALIZED
INFO [2020-07-03 10:31:05.404812] (MonitorAgent_1): INITIALIZED
[2020-07-03 10:31:05.463064] (SineGeneratorAgent_1): Connected output module:↵
↵MathAgent_1
[2020-07-03 10:31:05.468906] (SineGeneratorAgent_1): Connected output module:↵
↵MultiMathAgent_1
[2020-07-03 10:31:05.474625] (SineGeneratorAgent_1): Connected output module:↵
↵MonitorAgent_1

* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```



---

## Tutorial 3 - An advanced pipeline with multichannel signals.

---

We can use different channels for the receiver to handle specifically each channel name. This can be useful for example in splitting train and test channels in machine learning. Then, the user will need to implement specific handling of each channel in the receiving agent.

In this example, the *MultiGeneratorAgent* is used to send two different types of data - Sine and Cosine generator. This is done via specifying `send_output (channel="sine")` and `send_output (channel="cosine")`.

Then on the receiving end, the `on_received_message()` function checks for `message['channel']` to handle it separately.

Note that by default, *MonitorAgent* is only subscribed to the "default" channel. Hence it will not respond to the "cosine" and "sine" channel.

```
[1]: # %load tutorial_3_multi_channel.py
from agentMET4FOF.agents import AgentMET4FOF, AgentNetwork, MonitorAgent
from agentMET4FOF.streams import SineGenerator, CosineGenerator

def minus(data, minus_val):
    return data-minus_val

def plus(data, plus_val):
    return data+plus_val

class MultiGeneratorAgent(AgentMET4FOF):
    def init_parameters(self):
        self.sine_stream = SineGenerator()
        self.cos_stream = CosineGenerator()

    def agent_loop(self):
        if self.current_state == "Running":
            sine_data = self.sine_stream.next_sample() #dictionary
            cosine_data = self.sine_stream.next_sample() #dictionary
```

(continues on next page)

(continued from previous page)

```

        self.send_output(sine_data['x'], channel="sine")
        self.send_output(cosine_data['x'], channel="cosine")

class MultiOutputMathAgent (AgentMET4FOF):
    def init_parameters(self, minus_param=0.5, plus_param=0.5):
        self.minus_param = minus_param
        self.plus_param = plus_param

    def on_received_message(self, message):
        """
        Checks for message['channel'] and handles them separately
        Acceptable channels are "cosine" and "sine"
        """
        if message['channel'] == "cosine":
            minus_data = minus(message['data'], self.minus_param)
            self.send_output({'cosine_minus': minus_data})
        elif message['channel'] == 'sine':
            plus_data = plus(message['data'], self.plus_param)
            self.send_output({'sine_plus': plus_data})

def main():
    # start agent network server
    agentNetwork = AgentNetwork()
    # init agents
    gen_agent = agentNetwork.add_agent(agentType=MultiGeneratorAgent)
    multi_math_agent = agentNetwork.add_agent(agentType=MultiOutputMathAgent)
    monitor_agent = agentNetwork.add_agent(agentType=MonitorAgent)
    # connect agents : We can connect multiple agents to any particular agent
    # However the agent needs to implement handling multiple inputs
    agentNetwork.bind_agents(gen_agent, multi_math_agent)
    agentNetwork.bind_agents(gen_agent, monitor_agent)
    agentNetwork.bind_agents(multi_math_agent, monitor_agent)
    # set all agents states to "Running"
    agentNetwork.set_running_state()

    # allow for shutting down the network after execution
    return agentNetwork

if __name__ == '__main__':
    main()

```

```

Starting NameServer...
Broadcast server running on 0.0.0.0:9091
NS running on 127.0.0.1:3333 (127.0.0.1)
URI = PYRO:Pyro.NameServer@127.0.0.1:3333
INFO [2020-07-03 10:31:35.657368] (AgentController): INITIALIZED
INFO [2020-07-03 10:31:35.837487] (MultiGeneratorAgent_1): INITIALIZED
* Serving Flask app "agentMET4FOF.dashboard.Dashboard" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.

```

(continues on next page)

(continued from previous page)

```
    Use a production WSGI server instead.
* Debug mode: off
INFO [2020-07-03 10:31:35.921903] (MultiOutputMathAgent_1): INITIALIZED
INFO [2020-07-03 10:31:35.993695] (MonitorAgent_1): INITIALIZED
[2020-07-03 10:31:36.051394] (MultiGeneratorAgent_1): Connected output module:↵
↵MultiOutputMathAgent_1
[2020-07-03 10:31:36.057693] (MultiGeneratorAgent_1): Connected output module:↵
↵MonitorAgent_1
[2020-07-03 10:31:36.066761] (MultiOutputMathAgent_1): Connected output module:↵
↵MonitorAgent_1
SET STATE:    Running
* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```





## agentMET4FOF agents

```
class agentMET4FOF.agents.AgentMET4FOF (name="", host=None, serializer=None, trans-  
port=None, attributes=None)
```

Base class for all agents with specific functions to be overridden/supplied by user.

Behavioral functions for users to provide are `init_parameters`, `agent_loop` and `on_received_message`. Communicative functions are `bind_output`, `unbind_output` and `send_output`.

**agent\_loop** ()

User defined method for the agent to execute for *loop\_wait* seconds specified either in *self.loop\_wait* or explicitly via `init_agent_loop(loop_wait)`

To start a new loop, call `init_agent_loop(loop_wait)` on the agent Example of usage is to check the *current\_state* of the agent and send data periodically

**before\_loop** ()

This action is executed before initiating the loop

**bind\_output** (*output\_agent*)

Forms Output connection with another agent. Any call on `send_output` will reach this newly binded agent

Adds the agent to its list of Outputs.

**Parameters** `output_agent` (`AgentMET4FOF` or *list*) – Agent(s) to be binded to this agent's output channel

**handle\_process\_data** (*message*)

Internal method to handle incoming message before calling user-defined `on_received_message` method.

If *current\_state* is either `Stop` or `Reset`, it will terminate early before entering `on_received_message`

**init\_agent\_loop** (*loop\_wait: Optional[int] = 1.0*)

Initiates the agent loop, which iterates every *loop\_wait* seconds

Stops every timers and initiate a new loop.

**Parameters** `loop_wait` (*int*, *optional*) – The wait between each iteration of the loop

**init\_parameters** ()

User provided function to initialize parameters of choice.

**log\_info** (*message*)

Prints logs to be saved into logfile with Logger Agent

**Parameters** **message** (*str*) – Message to be logged to the internal Logger Agent

**on\_init** ()

Internal initialization to setup the agent: mainly on setting the dictionary of Inputs, Outputs, PubAddr.

Calls user-defined *init\_parameters()* upon finishing.

**Inputs**

Dictionary of Agents connected to its input channels. Messages will arrive from agents in this dictionary. Automatically updated when *bind\_output()* function is called

**Type** dict

**Outputs**

Dictionary of Agents connected to its output channels. Messages will be sent to agents in this dictionary. Automatically updated when *bind\_output()* function is called

**Type** dict

**PubAddr\_alias**

Name of Publish address socket

**Type** str

**PubAddr**

Publish address socket handle

**Type** str

**AgentType**

Name of class

**Type** str

**current\_state**

Current state of agent. Can be used to define different states of operation such as “Running”, “Idle”, “Stop”, etc.. Users will need to define their own flow of handling each type of *self.current\_state* in the *agent\_loop*

**Type** str

**loop\_wait**

The interval to wait between loop. Call *init\_agent\_loop* to restart the timer or set the value of *loop\_wait* in *init\_parameters* when necessary.

**Type** int

**memory\_buffer\_size**

The total number of elements to be stored in the agent *memory* When total elements exceeds this number, the latest elements will be replaced with the incoming data elements

**Type** int

**on\_received\_message** (*message*)

User-defined method and is triggered to handle the message passed by Input.

**Parameters** **message** (*Dictionary*) – The message received is in form  
{‘from’:agent\_name, ‘data’: data, ‘senderType’: agent\_class, ‘channel’:channel\_name}  
agent\_name is the name of the Input agent which sent the message data is the actual content of the message

**pack\_data** (*data, channel='default'*)

Internal method to pack the data content into a dictionary before sending out.

Special case : if the *data* is already a *message*, then the *from* and *senderType* will be altered to this agent, without altering the *data* and *channel* within the message this is used for more succinct data processing

and passing.

#### Parameters

- **data** (*argument*) – Data content to be packed before sending out to agents.
- **channel** (*str*) – Key of dictionary which stores data

#### Returns Packed message data

**Return type** dict of the form {'from':agent\_name, 'data': data, 'senderType': agent\_class, 'channel':channel\_name}.

#### reset ()

This method will be called on all agents when the global *reset\_agents* is called by the AgentNetwork and when the Reset button is clicked on the dashboard.

Method to reset the agent's states and parameters. User can override this method to reset the specific parameters.

#### send\_output (data, channel='default')

Sends message data to all connected agents in self.Outputs.

Output connection can first be formed by calling *bind\_output*. By default calls *pack\_data*(data) before sending out. Can specify specific channel as opposed to 'default' channel.

#### Parameters

- **data** (*argument*) – Data content to be sent out
- **channel** (*str*) – Key of *message* dictionary which stores data

#### Returns message

**Return type** dict of the form {'from':agent\_name, 'data': data, 'senderType': agent\_class, 'channel':channel\_name}.

#### send\_plot (fig: Union[matplotlib.figure.Figure, Dict[str, matplotlib.figure.Figure]], mode: str = 'image')

Sends plot to agents connected to this agent's Output channel.

This method is different from *send\_output* which will be sent to through the 'plot' channel to be handled.

Tradeoffs between "image" and "plotly" modes are that "image" are more stable and "plotly" are interactive. Note not all (complicated) matplotlib figures can be converted into a plotly figure.

#### Parameters

- **fig** (*matplotlib.figure.Figure* or *dict of matplotlib.figure.Figure*) – Alternatively, multiple figures can be nested in a dict (with any preferred keys) e.g {"Temperature":matplotlib.Figure, "Acceleration":matplotlib.Figure}
- **mode** (*str*) – "image" - converts into image via encoding at base64 string. "plotly" - converts into plotly figure using *mpl\_to\_plotly* Default: "image"

#### Returns graph

**Return type** str or plotly figure or dict of one of those converted figure(s)

#### stop\_agent\_loop ()

Stops agent\_loop from running. Note that the agent will still be responding to messages

#### unbind\_output (output\_agent)

Remove existing output connection with another agent. This reverses the *bind\_output* method

**Parameters** **output\_agent** (*AgentMET4FOF*) – Agent binded to this agent's output channel

**update\_data\_memory** (*agent\_from*, *data=None*)

Updates data stored in *self.memory* with the received message

Checks if sender agent has sent any message before If it did, then append, otherwise create new entry for it

#### Parameters

- **agent\_from** (*dict* / *str*) – if type is dict, we expect it to be the agentMET4FOF dict message to be compliant with older code otherwise, we expect it to be name of agent sender and *data* will need to be passed as parameter
- **data** – optional if *agent\_from* is a dict. Otherwise this parameter is compulsory. Any supported data which can be stored in dict as buffer.

```
class agentMET4FOF.agents.AgentNetwork (ip_addr='127.0.0.1', port=3333, connect=False, log_filename='log_file.csv', dashboard_modules=True, dashboard_extensions=[], dashboard_update_interval=3, dashboard_max_monitors=10, dashboard_port=8050)
```

Object for starting a new Agent Network or connect to an existing Agent Network specified by ip & port

Provides function to add agents, (un)bind agents, query agent network state, set global agent states Interfaces with an internal *\_AgentController* which is hidden from user

```
add_agent (name=' ', agentType=<class 'agentMET4FOF.agents.AgentMET4FOF'>, log_mode=True, memory_buffer_size=1000000, ip_addr=None)
```

Instantiates a new agent in the network.

#### Parameters

- **str** (*name*) – with the same name. Defaults to the agent's class name.
- **AgentMET4FOF** (*agentType*) – network. Defaults to *AgentMET4FOF*
- **bool** (*log\_mode*) – Logger Agent. Defaults to *True*.

#### Returns AgentMET4FOF

**Return type** Newly instantiated agent

```
agents (filter_agent=None)
```

Returns all agent names connected to Agent Network.

#### Returns list

**Return type** names of all agents

```
bind_agents (source, target)
```

Binds two agents communication channel in a unidirectional manner from *source* Agent to *target* Agent

Any subsequent calls of *source.send\_output()* will reach *target* Agent's message queue.

#### Parameters

- **source** (*AgentMET4FOF*) – Source agent whose Output channel will be binded to *target*
- **target** (*AgentMET4FOF*) – Target agent whose Input channel will be binded to *source*

```
connect (ip_addr='127.0.0.1', port=3333, verbose=True)
```

#### Parameters

- **ip\_addr** (*str*) – IP Address of server to connect to
- **port** (*int*) – Port of server to connect to

**get\_agent** (*agent\_name*)

Returns a particular agent connected to Agent Network.

**Parameters** **agent\_name** (*str*) – Name of agent to search for in the network

**set\_agents\_state** (*filter\_agent=None, state='Idle'*)

Blanket operation on all agents to set their *current\_state* attribute to given state

Can be used to define different states of operation such as “Running”, “Idle”, “Stop”, etc.. Users will need to define their own flow of handling each type of *self.current\_state* in the *agent\_loop*

**Parameters**

- **filter\_agent** (*str*) – (Optional) Filter name of agents to set the states
- **state** (*str*) – State of agents to set

**set\_running\_state** (*filter\_agent=None*)

Blanket operation on all agents to set their *current\_state* attribute to “Running”

Users will need to define their own flow of handling each type of *self.current\_state* in the *agent\_loop*

**Parameters** **filter\_agent** (*str*) – (Optional) Filter name of agents to set the states

**set\_stop\_state** (*filter\_agent=None*)

Blanket operation on all agents to set their *current\_state* attribute to “Stop”

Users will need to define their own flow of handling each type of *self.current\_state* in the *agent\_loop*

**Parameters** **filter\_agent** (*str*) – (Optional) Filter name of agents to set the states

**shutdown** ()

Shutowns the entire agent network and all agents

**start\_server** (*ip\_addr='127.0.0.1', port=3333*)

**Parameters**

- **ip\_addr** (*str*) – IP Address of server to start
- **port** (*int*) – Port of server to start

**unbind\_agents** (*source, target*)

Unbinds two agents communication channel in a unidirectional manner from *source* Agent to *target* Agent

This is the reverse of *bind\_agents()*

**Parameters**

- **source** (*AgentMET4FOF*) – Source agent whose Output channel will be unbinded from *target*
- **target** (*AgentMET4FOF*) – Target agent whose Input channel will be unbinded from *source*

**class** agentMET4FOF.agents.**DataStreamAgent** (*name="", host=None, serializer=None, transport=None, attributes=None*)

Able to simulate generation of datastream by loading a given DataStreamMET4FOF object.

Can be used in incremental training or batch training mode. To simulate batch training mode, set *pretrain\_size=-1*, otherwise, set *pretrain\_size* and *batch\_size* for the respective See *DataStreamMET4FOF* on loading your own data set as a data stream.

**agent\_loop** ()

User defined method for the agent to execute for *loop\_wait* seconds specified either in *self.loop\_wait* or explicitly via `'init_agent_loop(loop_wait)'`

To start a new loop, call `init_agent_loop(loop_wait)` on the agent. Example of usage is to check the `current_state` of the agent and send data periodically

```
init_parameters (stream=<agentMET4FOF.streams.DataStreamMET4FOF object>, pre-  
train_size=None, batch_size=1, loop_wait=1, randomize=False)
```

#### Parameters

- **stream** (`DataStreamMET4FOF`) – A `DataStreamMET4FOF` object which provides the sample data
- **pretrain\_size** (*int*) – The number of sample data to send through in the first loop cycle, and subsequently, the `batch_size` will be used
- **batch\_size** (*int*) – The number of sample data to send in every loop cycle
- **loop\_wait** (*int*) – The duration to wait (seconds) at the end of each loop cycle before going into the next cycle
- **randomize** (*bool*) – Determines if the dataset should be shuffled before streaming

#### **reset** ()

This method will be called on all agents when the global `reset_agents` is called by the AgentNetwork and when the Reset button is clicked on the dashboard.

Method to reset the agent's states and parameters. User can override this method to reset the specific parameters.

```
class agentMET4FOF.agents.MonitorAgent (name="", host=None, serializer=None, trans-  
port=None, attributes=None)
```

Unique Agent for storing plots and data from messages received from input agents.

The dashboard searches for Monitor Agents' `memory` and `plots` to draw the graphs "plot" channel is used to receive base64 images from agents to plot on dashboard

#### **memory**

Dictionary of format `{agent1_name : agent1_data, agent2_name : agent2_data}`

**Type** dict

#### **plots**

Dictionary of format `{agent1_name : agent1_plot, agent2_name : agent2_plot}`

**Type** dict

#### **plot\_filter**

List of keys to filter the 'data' upon receiving message to be saved into memory. Used to specifically select only a few keys to be plotted

**Type** list of str

```
init_parameters (plot_filter=[], custom_plot_function=-1, **kwargs)
```

User provided function to initialize parameters of choice.

```
on_received_message (message)
```

Handles incoming data from 'default' and 'plot' channels.

Stores 'default' data into `self.memory` and 'plot' data into `self.plots`

**Parameters** **message** (*dict*) – Acceptable channel values are 'default' or 'plot'

#### **reset** ()

This method will be called on all agents when the global `reset_agents` is called by the AgentNetwork and when the Reset button is clicked on the dashboard.

Method to reset the agent's states and parameters. User can override this method to reset the specific parameters.

**update\_plot\_memory** (*message*)

Updates plot figures stored in *self.plots* with the received message

**Parameters** **message** (*dict*) – Standard message format specified by AgentMET4FOF class

Message['data'] needs to be base64 image string and can be nested in dictionary for multiple plots Only the latest plot will be shown kept and does not keep a history of the plots.





## agentMET4FOF streams

```
class agentMET4FOF.streams.CosineGenerator (num_cycles=1000)
```

```
class agentMET4FOF.streams.DataStreamMET4FOF
```

Class for creating finite datastream for ML with  $x$  as inputs and  $y$  as target Data can be fetched sequentially using *next\_sample()* or all at once *all\_samples()*

For sensors data: The format shape for 2D data stream (num\_samples, n\_sensors) The format shape for 3D data stream (num\_samples, sample\_length , n\_sensors)

```
all_samples ()
```

Returns all the samples in the data stream

**Returns samples**

**Return type** dict of the form  $\{ 'x': \text{current\_sample\_x}, 'y': \text{current\_sample\_y} \}$

```
next_sample (batch_size=1)
```

Fetches the samples from the data stream and advances the internal pointer *current\_idx*

**Parameters** **batch\_size** (*int*) – number of batches to get from data stream

**Returns samples**

**Return type** dict of the form  $\{ 'x': \text{current\_sample\_x}, 'y': \text{current\_sample\_y} \}$

```
class agentMET4FOF.streams.SineGenerator (num_cycles=1000)
```

```
agentMET4FOF.streams.extract_x_y (message)
```

**Extracts features & target from *message['data']* with expected structure such as :**

1. tuple - (x,y)
2. dict -  $\{ 'x': \text{x\_data}, 'y': \text{y\_data} \}$

Handle data structures of dictionary to extract features & target



## agentMET4FOF dashboard

```
class agentMET4FOF.dashboard.Dashboard.AgentDashboard(dashboard_modules=[],
                                                    dashboard_layouts=[], dash-
                                                    board_update_interval=3,
                                                    max_monitors=10,
                                                    ip_addr='127.0.0.1',
                                                    port=8050,           agent-
                                                    Network='127.0.0.1',
                                                    agent_ip_addr=3333,
                                                    agent_port=None)
```

Class for the web dashboard which runs with the AgentNetwork object, which by default are on the same IP. Optional to run the dashboard on a separate IP by providing the right parameters. See example for an implementation of a separate run of dashboard to connect to an existing agent network. If there is no existing agent network, error will show up. An internal `_Dashboard_Control` object is instantiated inside this object, which manages access to the AgentNetwork.

```
init_app_layout (update_interval_seconds=3, max_monitors=10, dashboard_layouts=[])
```

Initialises the overall dash app “layout” which has two sub-pages (Agent network and ML experiment)

### Parameters

- **update\_interval\_seconds** (*float or int*) – Auto refresh rate which the app queries the states of Agent Network to update the graphs and display
- **max\_monitors** (*int*) – Due to complexity in managing and instantiating dynamic figures, a maximum number of monitors is specified first and only the each Monitor Agent will occupy one of these figures. It is not ideal, but will undergo changes for the better.

### Returns app

**Return type** Dash app object



## CHAPTER 8

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



## CHAPTER 9

---

### References

---





---

## Bibliography

---

- [Bang2019] Bang X. Yong, A. Brintrup Multi Agent System for Machine Learning Under Uncertainty in Cyber Physical Manufacturing System, 9th Workshop on Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future



### a

`agentMET4FOF.agents`, [21](#)  
`agentMET4FOF.dashboard.Dashboard`, [31](#)  
`agentMET4FOF.streams`, [29](#)



## A

`add_agent()` (*agentMET4FOF.agents.AgentNetwork* method), 24  
`agent_loop()` (*agentMET4FOF.agents.AgentMET4FOF* method), 21  
`agent_loop()` (*agentMET4FOF.agents.DataStreamAgent* method), 25  
`AgentDashboard` (class in *agentMET4FOF.dashboard.Dashboard*), 31  
`AgentMET4FOF` (class in *agentMET4FOF.agents*), 21  
`agentMET4FOF.agents` (module), 21  
`agentMET4FOF.dashboard.Dashboard` (module), 31  
`agentMET4FOF.streams` (module), 29  
`AgentNetwork` (class in *agentMET4FOF.agents*), 24  
`agents()` (*agentMET4FOF.agents.AgentNetwork* method), 24  
`AgentType` (*agentMET4FOF.agents.AgentMET4FOF* attribute), 22  
`all_samples()` (*agentMET4FOF.streams.DataStreamMET4FOF* method), 29

## B

`before_loop()` (*agentMET4FOF.agents.AgentMET4FOF* method), 21  
`bind_agents()` (*agentMET4FOF.agents.AgentNetwork* method), 24  
`bind_output()` (*agentMET4FOF.agents.AgentMET4FOF* method), 21

## C

`connect()` (*agentMET4FOF.agents.AgentNetwork* method), 24

`CosineGenerator` (class in *agentMET4FOF.streams*), 29

`current_state` (*agentMET4FOF.agents.AgentMET4FOF* attribute), 22

## D

`DataStreamAgent` (class in *agentMET4FOF.agents*), 25

`DataStreamMET4FOF` (class in *agentMET4FOF.streams*), 29

## E

`extract_x_y()` (in module *agentMET4FOF.streams*), 29

## G

`get_agent()` (*agentMET4FOF.agents.AgentNetwork* method), 24

## H

`handle_process_data()` (*agentMET4FOF.agents.AgentMET4FOF* method), 21

## I

`init_agent_loop()` (*agentMET4FOF.agents.AgentMET4FOF* method), 21

`init_app_layout()` (*agentMET4FOF.dashboard.Dashboard.AgentDashboard* method), 31

`init_parameters()` (*agentMET4FOF.agents.AgentMET4FOF* method), 21

`init_parameters()` (*agentMET4FOF.agents.DataStreamAgent* method), 26

`init_parameters()` (*agentMET4FOF.agents.MonitorAgent method*), 26

`Inputs` (*agentMET4FOF.agents.AgentMET4FOF attribute*), 22

## L

`log_info()` (*agentMET4FOF.agents.AgentMET4FOF method*), 21

`loop_wait` (*agentMET4FOF.agents.AgentMET4FOF attribute*), 22

## M

`memory` (*agentMET4FOF.agents.MonitorAgent attribute*), 26

`memory_buffer_size` (*agentMET4FOF.agents.AgentMET4FOF attribute*), 22

`MonitorAgent` (*class in agentMET4FOF.agents*), 26

## N

`next_sample()` (*agentMET4FOF.streams.DataStreamMET4FOF method*), 29

## O

`on_init()` (*agentMET4FOF.agents.AgentMET4FOF method*), 22

`on_received_message()` (*agentMET4FOF.agents.AgentMET4FOF method*), 22

`on_received_message()` (*agentMET4FOF.agents.MonitorAgent method*), 26

`Outputs` (*agentMET4FOF.agents.AgentMET4FOF attribute*), 22

## P

`pack_data()` (*agentMET4FOF.agents.AgentMET4FOF method*), 22

`plot_filter` (*agentMET4FOF.agents.MonitorAgent attribute*), 26

`plots` (*agentMET4FOF.agents.MonitorAgent attribute*), 26

`PubAddr` (*agentMET4FOF.agents.AgentMET4FOF attribute*), 22

`PubAddr_alias` (*agentMET4FOF.agents.AgentMET4FOF attribute*), 22

## R

`reset()` (*agentMET4FOF.agents.AgentMET4FOF method*), 23

`reset()` (*agentMET4FOF.agents.DataStreamAgent method*), 26

`reset()` (*agentMET4FOF.agents.MonitorAgent method*), 26

## S

`send_output()` (*agentMET4FOF.agents.AgentMET4FOF method*), 23

`send_plot()` (*agentMET4FOF.agents.AgentMET4FOF method*), 23

`set_agents_state()` (*agentMET4FOF.agents.AgentNetwork method*), 25

`set_running_state()` (*agentMET4FOF.agents.AgentNetwork method*), 25

`set_stop_state()` (*agentMET4FOF.agents.AgentNetwork method*), 25

`shutdown()` (*agentMET4FOF.agents.AgentNetwork method*), 25

`SineGenerator` (*class in agentMET4FOF.streams*), 29

`start_server()` (*agentMET4FOF.agents.AgentNetwork method*), 25

`stop_agent_loop()` (*agentMET4FOF.agents.AgentMET4FOF method*), 23

## U

`unbind_agents()` (*agentMET4FOF.agents.AgentNetwork method*), 25

`unbind_output()` (*agentMET4FOF.agents.AgentMET4FOF method*), 23

`update_data_memory()` (*agentMET4FOF.agents.AgentMET4FOF method*), 23

`update_plot_memory()` (*agentMET4FOF.agents.MonitorAgent method*), 27