agentMET4FOF Documentation

Bang Xiang Yong

Apr 28, 2021

Getting started:

CHAPTER 1

Table of content

CHAPTER 2

Quickstart

agentMET4FOF comes bundled with some tutorials to get you started as quick as possible. In your Python console execute the following to run the first tutorial.

```
>>> from agentMET4FOF_tutorials.tutorial_1_generator_agent import demonstrate_
↪generator_agent_use
>>> generator_agent_network = 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

------------------------------------------------------------
|                                                          |
| Your agent network is starting up. Open your browser and |
| visit the agentMET4FOF dashboard on http://127.0.0.1:8050/ |
|                                                          |
------------------------------------------------------------

INFO [2021-02-05 18:12:52.277759] (SineGeneratorAgent_1): INITIALIZED
INFO [2021-02-05 18:12:52.302862] (MonitorAgent_1): INITIALIZED
[2021-02-05 18:12:52.324078] (SineGeneratorAgent_1): Connected output module:␣
↪MonitorAgent_1
SET STATE:   Running
[...]
```

```
>>> generator_agent_network.shutdown()
0
NS shut down.
```

CHAPTER 3

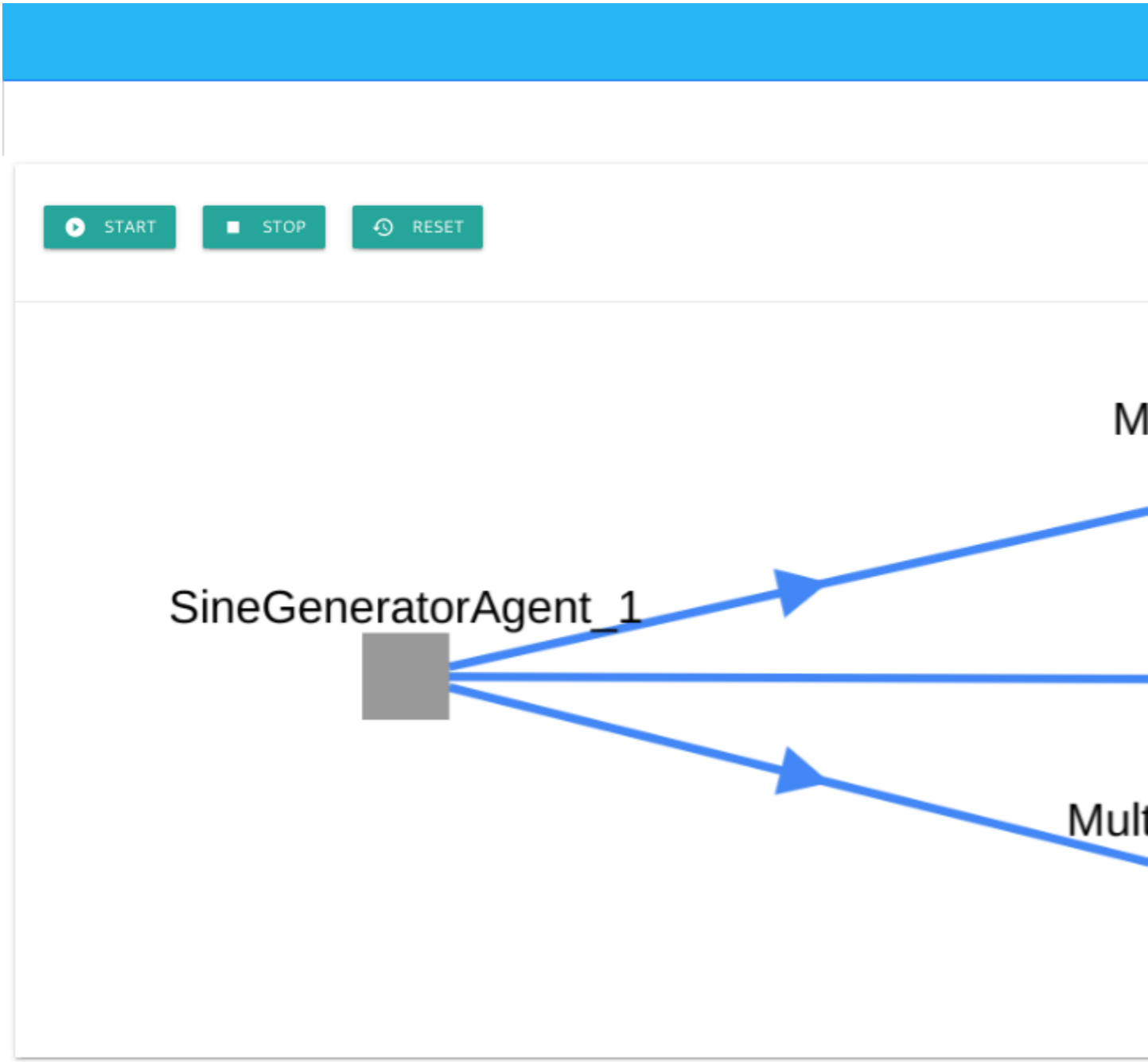
About

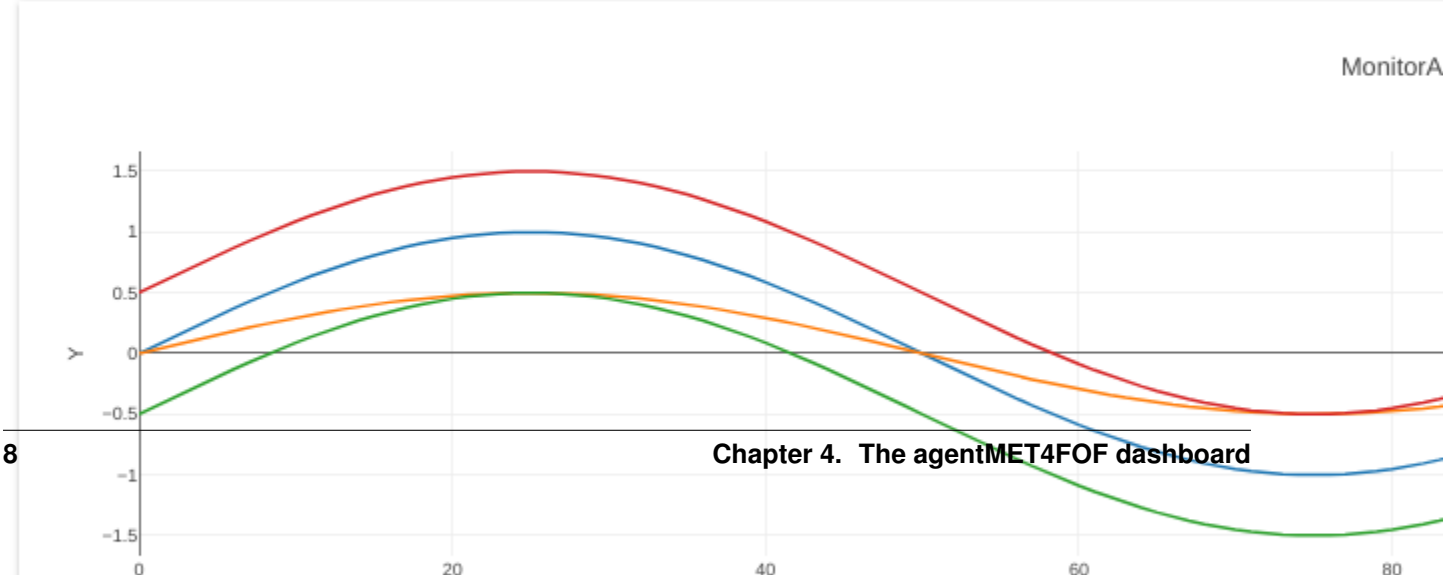
CHAPTER 4

The agentMET4FOF dashboard

agentMET4FOF comes bundled with our so called *dashboard*. It is an optional component of every agent network and provides a web browser based view. You can observe the state of your agents, modify the connections between them and even add more pre-made agents to your network all during run-time. The address to your dashboard is printed to the console on every launch of an agent network.

The following image is close to what you will find in your browser on execution of tutorial 2. For details on the tutorials visit our *video tutorial series*.


START
STOP
RESET
SineGeneratorAgent_1
M
Mult


MonitorA
1.5
1
0.5
0
−0.5
−1
−1.5
Y
0
20
40
60
80

Screenshot

CHAPTER 5

Documentation and video tutorials

Extended documentation can be found on ReadTheDocs.

5.1 Video tutorial series

Additionally, we provide some video tutorials based on agentMET4FOF 0.4.1 on the project homepage in the section *Tutorials for the multi-agent system agentMET4FOF*. You can self-register on the linked page and get started immediately. The video series begins with our motivation for creating agentMET4FOF, guide you through the installation of Python and other recommended software until you execute the tutorials on your machine.

5.2 Live online tutorial during early development

In an early development stage we held a live online tutorial based on agentMET4FOF 0.1.0 which you can download.

If questions arise, or you feel something is missing, reach out to us.

CHAPTER 6

Installation

The installation of agentMET4FOF is as straightforward as the Python ecosystem suggests. In the *video tutorials series* we guide you through every step until you have agentMET4FOF running on your machine. Besides that we have more details in the installation section of the docs.

CHAPTER 7

Coming soon

- Dockerize agentMET4FOF
- Improve handling of metadata
- Further improve plotting

For a comprehensive overview of current development activities and upcoming tasks, take a look at the project board, issues and pull requests.

CHAPTER 8

Citation

If you publish results obtained with the help of agentMET4FOF, please cite the linked DOI.

CHAPTER 9

Acknowledgement

This work was part of the Joint Research Project Metrology for the Factory of the Future (Met4FoF), project number 17IND12 of the European Metrology Programme for Innovation and Research (EMPIR). The EMPIR is jointly funded by the EMPIR participating countries within EURAMET and the European Union.

CHAPTER 10

Disclaimer

This software is developed as a joint effort of several project partners namely:

- Institute for Manufacturing of the University of Cambridge (IfM)
- Physikalisch-Technische Bundesanstalt (PTB)
- Van Swinden Laboratory (VSL)
- National Physics Laboratory (NPL)

under the lead of IfM. The software is made available “as is” free of cost. The authors and their institutions assume no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, safety, suitability or any other characteristic. In no event will the authors be liable for any direct, indirect or consequential damage arising in connection with the use of this software.

CHAPTER 11

©License

agentMET4FOF is distributed under the LGPLv3 license.

CHAPTER 12

Installation of agentMET4FOF

The installation of agentMET4FOF is as straightforward as the Python ecosystem suggests. In the video tutorial series linked in the README we guide you through every step until you have agentMET4FOF running on your machine.

If you want to take the steps manually we guide you through in this document.

12.1 Set up a virtual environment

For the motivation of creating to virtual environment for your installation of the agents check the official Python docs on that topic. You have the option to do this with *Anaconda*, if you already have it installed, or use the Python built-in tool `venv`. The commands differ slightly between *Windows* and *Mac/Linux*.

12.1.1 Create a `venv` Python environment on Windows

In your Windows PowerShell execute the following to set up a virtual environment in a folder of your choice.

```
PS C:> cd C:\LOCAL\PATH\TO\ENVS
PS C:\LOCAL\PATH\TO\ENVS> py -3 -m venv agentMET4FOF_venv
PS C:\LOCAL\PATH\TO\ENVS> agentMET4FOF_venv\Scripts\activate
```

Proceed to *step 2*.

12.1.2 Create a `venv` Python environment on Mac & Linux

In your terminal execute the following to set up a virtual environment in a folder of your choice.

```
$ cd /LOCAL/PATH/TO/ENVS
$ python3 -m venv agentMET4FOF_venv
$ source agentMET4FOF_venv/bin/activate
```

Proceed to *step 2*.

12.1.3 Create an Anaconda Python environment

To get started with your present *Anaconda* installation just go to *Anaconda prompt* and execute

```
$ cd /LOCAL/PATH/TO/ENVS
$ conda env create --file /LOCAL/PATH/TO/agentMET4FOF/environment.yml
```

That's it!

12.2 Install agentMET4FOF via `pip`

Once you activated your virtual environment, you can install agentMET4FOF via:

```
pip install agentMET4FOF
```

```
Collecting agentMET4FOF
[...]
Successfully installed agentMET4FOF-[...] [...]
```

That's it!

12.3 Get started developing

As a starter we recommend working through the tutorials which we present in detail in our video tutorial series linked in the README.

12.4 Orphaned processes

In the event of agents not terminating cleanly, you can end all Python processes running on your system (caution: the following commands affect **all** running Python processes, not just those that emerged from the agents).

12.4.1 Killing all Python processes in Windows

In your Windows command prompt execute the following to terminate all python processes.

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

12.4.2 Killing all Python processes on Mac and Linux

In your terminal execute the following to terminate all python processes.

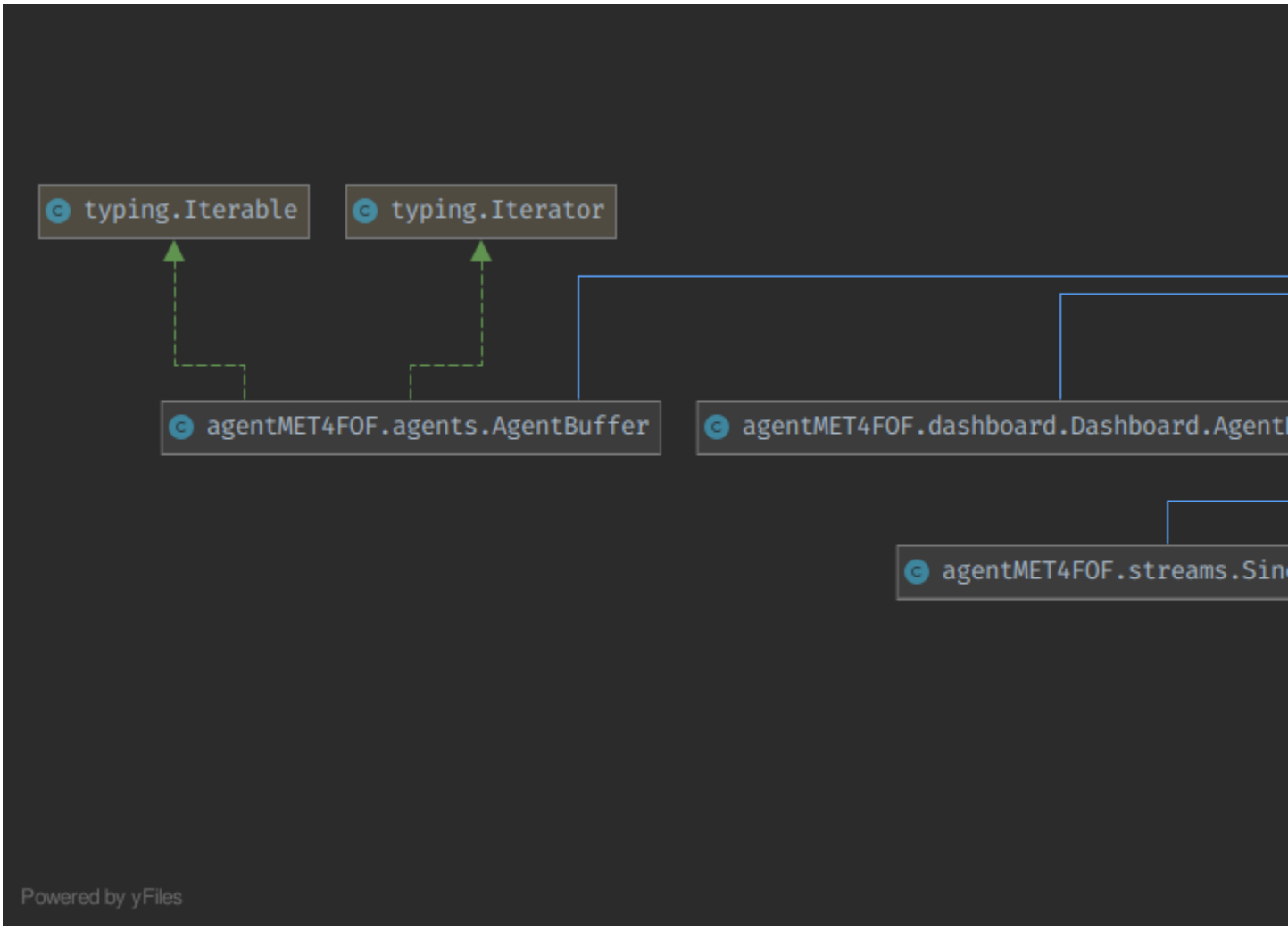
```
$ pkill python
$
```

CHAPTER 13

UML diagrams of agentMET4FOF

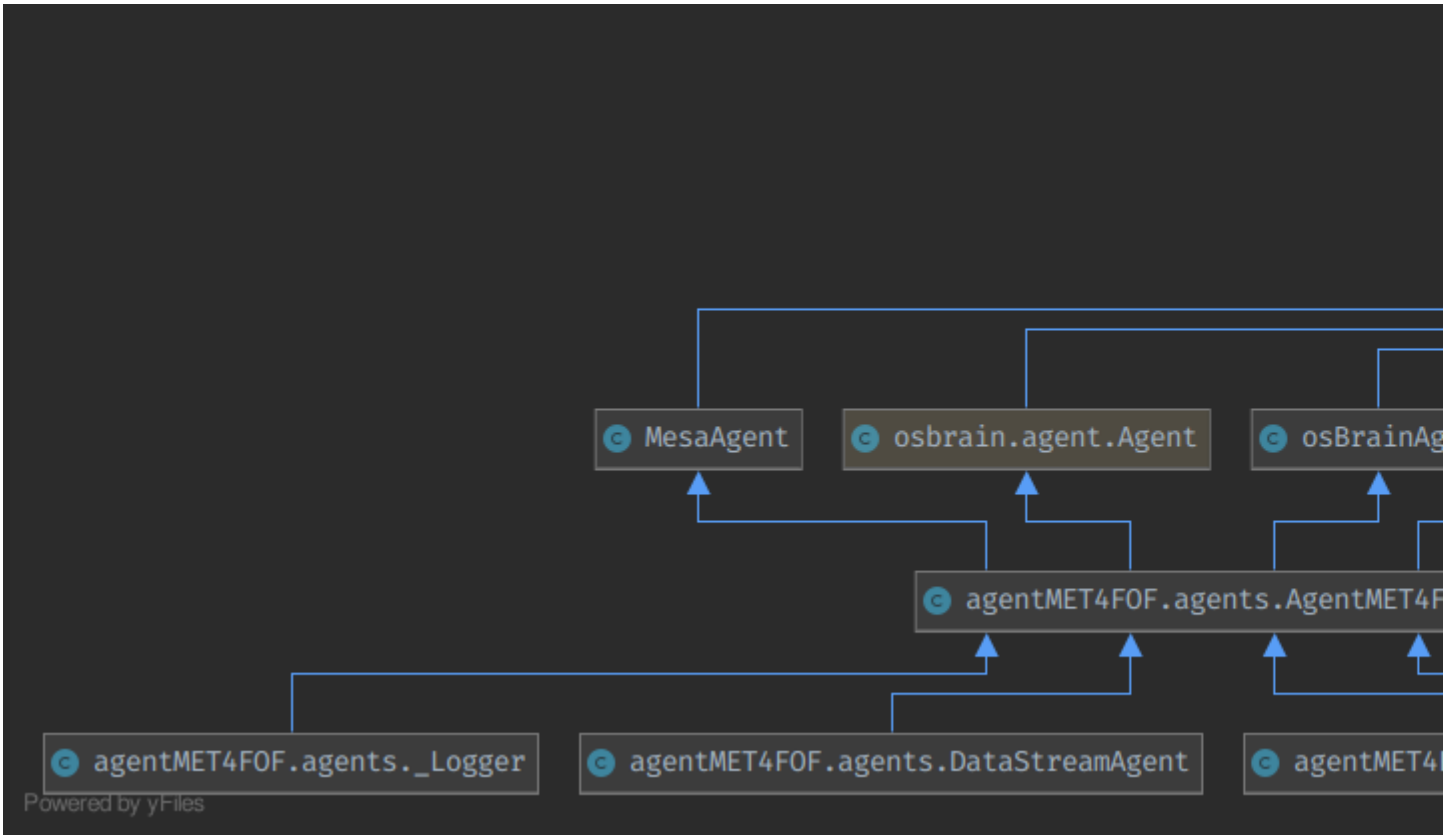
In this section UML class diagrams of almost all components of agentMET4FOF are listed and shown. The images are click- and zoomable in the browser but can be downloaded for further investigation via right-click.

13.1 Overview



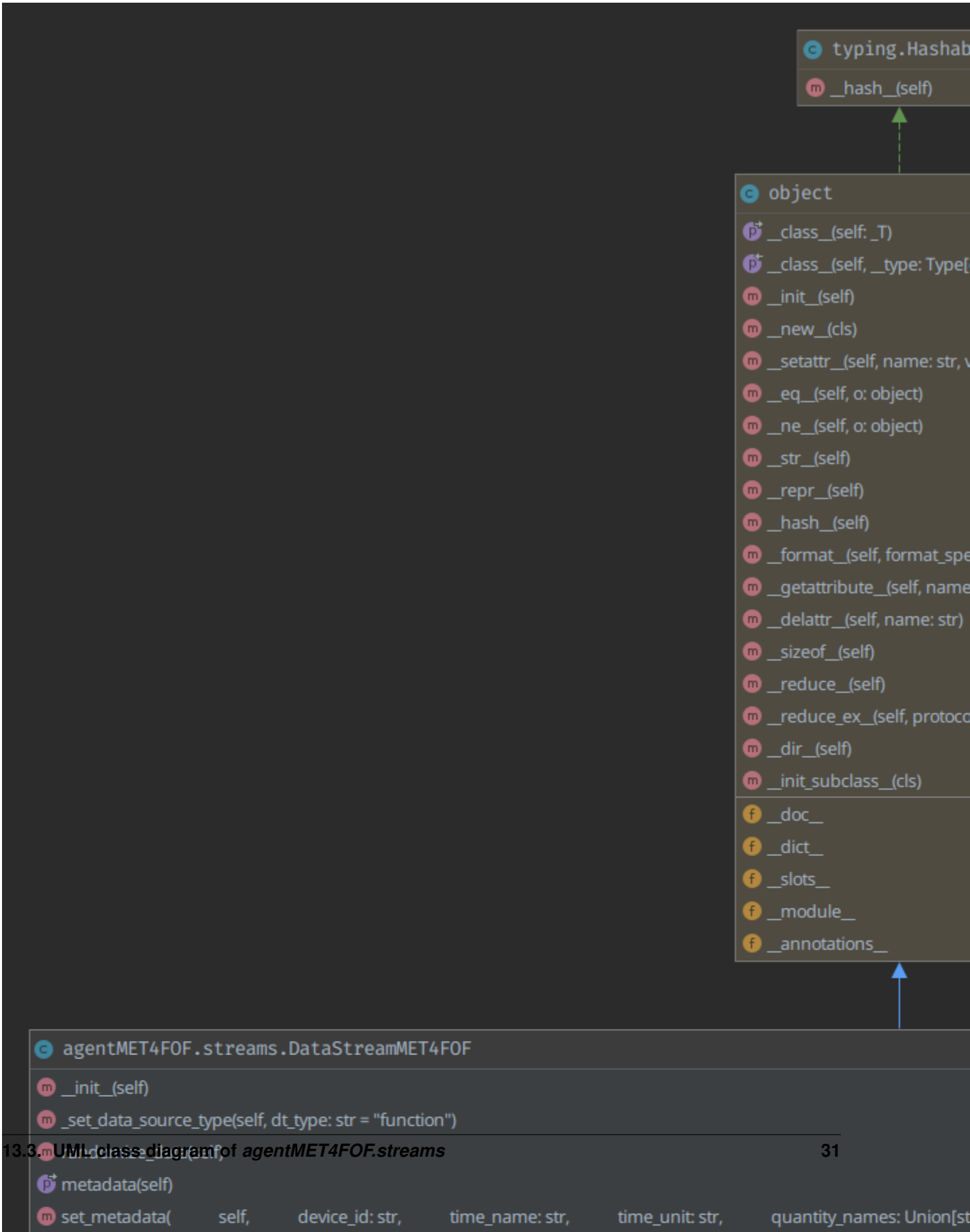
of all agentMET4FOF classes

13.2 UML class diagram of *agentMET4FOF.agents*

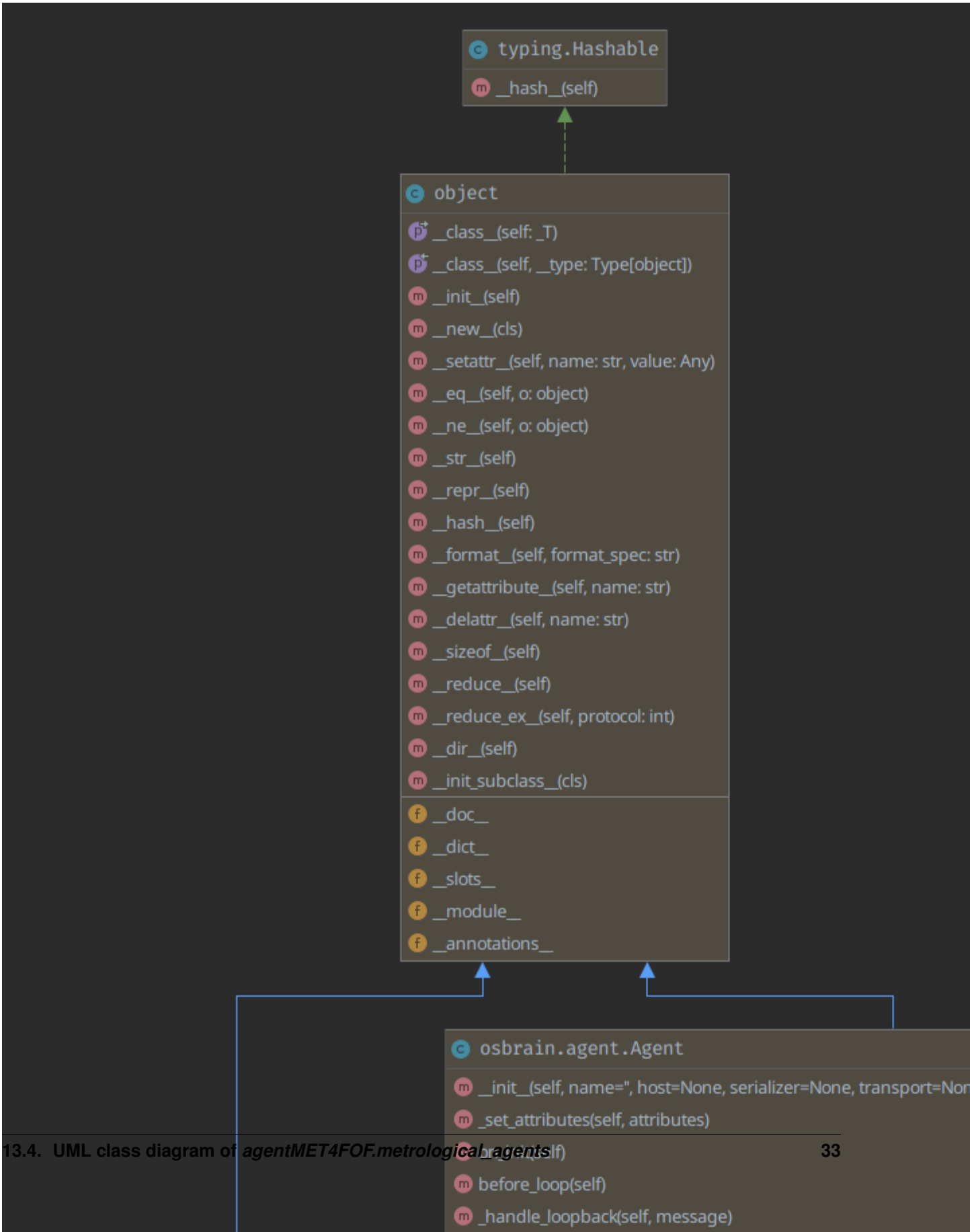


of classes in agentMETFOF.agents

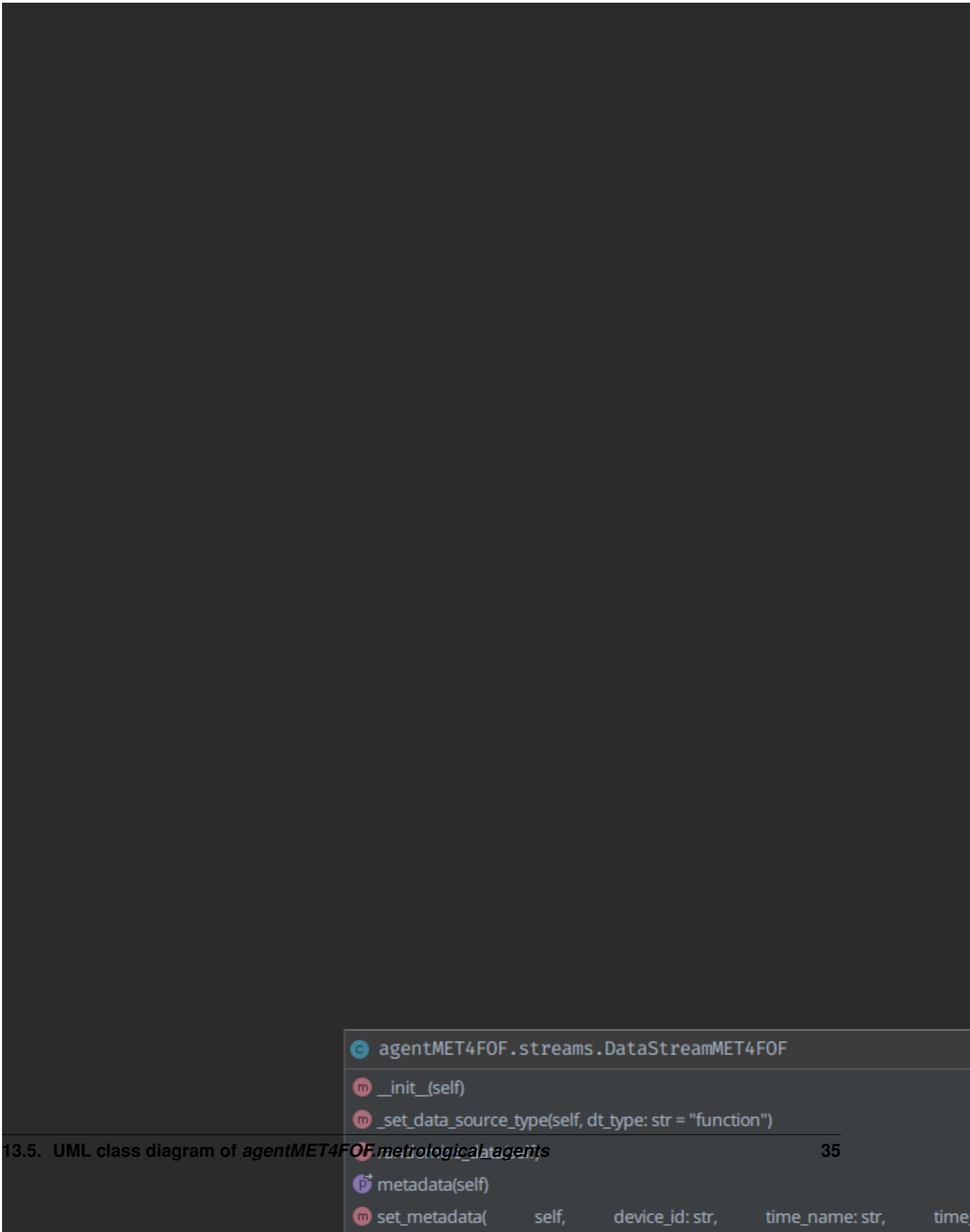
13.3 UML class diagram of *agentMET4FOF.streams*



13.4 UML class diagram of *agentMET4FOF.metrological_agents*



13.5 UML class diagram of *agentMET4FOF.metrological_agents*



13.6 Creation of the UMLs

We used *PyCharm Professional* to create the diagrams in "Hierarchic Group Layout".

Date of creation for all diagrams: 2021-02-03 on commit 54a4863.

CHAPTER 14

How to contribute to agentMET4FOF

If you want to contribute back to the project, we provide a guide to get the desired system configuration aligned with our development environments. The code you produce should be seamlessly integrable into agentMET4FOF by aligning your work with the established workflows. This guide should work on all platforms and provide everything needed to start developing for agentMET4FOF. Please open an issue or ideally contribute to this guide as a start, if problems or questions arise.

14.1 Guiding principles

The agentMET4FOF development process is based on the following guiding principles:

- actively maintain, ensuring security vulnerabilities or other issues are resolved in a timely manner
- employ state-of-the-art development practices and tools, specifically
  - follow semantic versioning
  - use conventional commit messages
  - consider the PEP8 style guide, wherever feasible

14.1.1 Get the code on GitHub and locally

For collaboration, we recommend forking the repository as described here. Simply apply the changes to your fork and open a Pull Request on GitHub as described here. For small changes it will be sufficient to just apply your changes on GitHub and send the PR right away. For more comprehensive work, you should clone your fork and read on carefully.

14.1.2 Initial development setup

This guide assumes you already have a valid runtime environment for agentMET4FOF as described in the Installation guide.

First install the known to work configuration of our dependencies into you virtual environment:

```
(agentMET4FOF_venv) $ pip install -r requirements.txt -r dev-requirements.txt
```

14.1.3 Advised toolset

If you followed the steps for the *initial development setup* you have everything at your hands:

- *Sphinx* for automated generation of our documentation on ReadTheDocs
- *pytest* as testing framework backed by *hypothesis* and *coverage*.
- *python-semantic-release* in
- our pipeline on *CircleCI* . All requirements for contributions are derived from this.

14.1.4 Coding style

As long as the readability of mathematical formulations is not impaired, our code shoulds follow PEP8. We know we can improve on this requirement for the existing code base as well, but all code added should already conform to PEP8. For automating this uniform formatting task we use the Python package *black*. It is easy to handle and integrable into most common IDEs, such that it is automatically applied.

14.1.5 Commit messages

agentMET4FOF commit messages follow some conventions to be easily human and machine-readable.

Commit message structure

Conventional commit messages are required for the following:

- Releasing automatically according to semantic versioning
- Generating a changelog automatically

Parts of the commit messages and links appear in the changelogs of subsequent releases as a result. We use the following types:

- *feat*: for commits that introduce new features (this correlates with MINOR in semantic versioning)
- *docs*: for commits that contribute significantly to documentation
- *fix*: commits in which bugs are fixed (this correlates with PATCH in semantic versioning)
- *test*: Commits that apply significant changes to tests
- *chore*: Commits that affect other non-PyDynamic components (e.g. ReadTheDocs, Git , . . .)
- *revert*: commits, which undo previous commits using `git revert`
- *wip*: Commits which are not recognizable as one of the above-mentioned types until later, usually during a PR merge. The merge commit is then marked as the corresponding type.

Of the types mentioned above, the following appear in separate sections of the changelog:

- *Feature*: *feat*
- *Documentation*: *docs*
- *Fix*: *fix*

- *Test*: *test*

Commit message styling

Based on established community standards, the first line of a commit message should complete the following sentence:

> If this commit is applied, it will...

More comprehensive messages should contain an empty line after that and everything else needed starting from the third line. Each line should not exceed 100 characters.

BREAKING CHANGEs

Since agentMET4FOF is not yet considered stable, we do not mark BREAKING CHANGES. As a consequence, at any time commits may changes parts of agentMET4FOF's public interface so that previously written code may no longer be executable. If this occurs we try though, to mention migration strategies in the corresponding release descriptions.

Commit message examples

For examples please checkout the Git Log.

14.1.6 Testing

We strive to increase our code coverage with every change introduced. This requires that every new feature and every change to existing features is accompanied by appropriate *pytest* testing. We test the basic components for correctness and, if necessary, the integration into the big picture. It is usually sufficient to create appropriately named methods in one of the existing modules in the subfolder test. If necessary add a new module that is appropriately named.

14.2 Workflow for adding completely new functionality

In case you add a new feature you generally follow the pattern:

- read through and follow this contribution advices and tips, especially regarding the *advised tool* set and *coding style*
- open an according issue to submit a feature request and get in touch with other agentMET4FOF developers and users
- fork the repository or update the *develop* branch of your fork and create an arbitrary named feature branch from *develop*
- decide which package and module your feature should be integrated into
- if there is no suitable package or module, create a new one and a corresponding module in the *tests* subdirectory with the same name prefixed by *test_*
- if new dependencies are introduced, add them to *setup.py* or *dev-requirements.in*
- during development write tests in alignment with existing test modules, for example *test_addremove_metrological_agents*
- write docstrings in the NumPy docstring format
- as early as possible create a draft pull request onto the upstream's *develop* branch

- once you think your changes are ready to merge, request a review from the *agentMET4FOF collaborators* (you will find them in the according drop-down) and mark your PR as *ready for review*
- at the latest now you will have the opportunity to review the documentation automatically generated from the docstrings on ReadTheDocs after your reviewers will set up everything
- resolve the conversations and have your pull request merged

14.3 Documentation

The documentation of agentMET4FOF consists of three parts. Every adaptation of an existing feature and every new feature requires adjustments on all three levels:

- user documentation on ReadTheDocs
- examples in the form of Jupyter notebooks for extensive features and Python scripts for features which can be comprehensively described with few lines of commented code
- developer documentation in the form of comments in the code

14.3.1 User documentation

To locally generate a preview of what ReadTheDocs will generate from your docstrings, you can simply execute after activating your virtual environment:

```
(agentMET4FOF_venv) $ sphinx-build docs/ docs/_build
Sphinx v3.1.1 in Verwendung
making output directory...
[...]
build abgeschlossen.

The HTML pages are in docs/_build.
```

After that you can open the file _./docs/*build/index.html* relative to the project's root with your favourite browser. Simply re-execute the above command after each change to the docstrings to update your local version of the documentation.

14.3.2 Examples

We want to provide extensive sample material for all agentMET4FOF features in order to simplify the use or even make it possible in the first place. We collect the examples in the subfolder agentMET4FOF_tutorials.

14.3.3 Comments in the code

Regarding comments in the code we recommend to invest 45 minutes for the PyCon DE 2019 Talk of Stefan Schwarzer, a 20+-years Python developer: Commenting code - beyond common wisdom.

14.4 Manage dependencies

We use *pip-tools* for dependency management. The root folder contains a *requirements.txt* and a *dev-requirements.txt* for the supported Python version. *pip-tools*' command `pip-compile` finds the right versions from the dependencies listed in *setup.py* and the *dev-requirements.in* and is manually run by the maintainers regularly.

14.5 Licensing

All contributions are released under agentMET4FOF's GNU Lesser General Public License v3.0.

CHAPTER 15

Changelog

15.1 v0.6.1 (2021-03-12)

15.1.1 Fix

- **metrological_streams:** Fixed computational error for batches of size more than one (`686bad5`)
- **agents:** Fixed `buffer.clear` method, which did not work anymore after moving from `memory` to `buffer`

15.1.2 Documentation

- **README:** Include the Zenodo DOI banner and add the Citation section (`4a57dd8`)
- **docstrings:** Seriously improve docstrings and type hinting in `agents.py` and `dashboard/LayoutHelper.py`

See all commits in this version

CHAPTER 16

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.
    """

    # The datatype of the stream will be SineGenerator.
    _sine_stream: SineGenerator
```

(continues on next page)

(continued from previous page)

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

        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["quantities"])


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

------------------------------------------------------------
|                                                          |
| Your agent network is starting up. Open your browser and |
| visit the agentMET4FOF dashboard on http://127.0.0.1:8050/ |
|                                                          |
------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
INFO [2021-02-05 19:13:45.925758] (SineGeneratorAgent_1): INITIALIZED
INFO [2021-02-05 19:13:45.960173] (MonitorAgent_1): INITIALIZED
[2021-02-05 19:13:45.975418] (SineGeneratorAgent_1): Connected output module:␣
↪MonitorAgent_1
SET STATE:   Running
[2021-02-05 19:13:46.932922] (SineGeneratorAgent_1): Pack time: 0.000957
[2021-02-05 19:13:46.935938] (SineGeneratorAgent_1): Sending: [0.]
[2021-02-05 19:13:46.939035] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.]), 'senderType': 'SineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:13:46.941337] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.])}
[2021-02-05 19:13:46.942394] (MonitorAgent_1): Tproc: 0.002278
[2021-02-05 19:13:47.932181] (SineGeneratorAgent_1): Pack time: 0.000614
[2021-02-05 19:13:47.935312] (SineGeneratorAgent_1): Sending: [0.06279052]
[2021-02-05 19:13:47.936553] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.06279052]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:13:47.941367] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052])}
[2021-02-05 19:13:47.942190] (MonitorAgent_1): Tproc: 0.004357
[2021-02-05 19:13:48.931650] (SineGeneratorAgent_1): Pack time: 0.00047
[2021-02-05 19:13:48.933397] (SineGeneratorAgent_1): Sending: [0.12533323]
[2021-02-05 19:13:48.934297] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.12533323]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:13:48.936482] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323])}
[2021-02-05 19:13:48.936997] (MonitorAgent_1): Tproc: 0.002201
[2021-02-05 19:13:49.932143] (SineGeneratorAgent_1): Pack time: 0.000937
[2021-02-05 19:13:49.940442] (SineGeneratorAgent_1): Sending: [0.18738131]
[2021-02-05 19:13:49.934471] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.18738131]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:13:49.938767] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131])}
[2021-02-05 19:13:49.939977] (MonitorAgent_1): Tproc: 0.004969
[2021-02-05 19:13:50.930904] (SineGeneratorAgent_1): Pack time: 0.000255
[2021-02-05 19:13:50.931636] (SineGeneratorAgent_1): Sending: [0.24868989]
[2021-02-05 19:13:50.932383] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.24868989]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:13:50.933944] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989])}
[2021-02-05 19:13:50.934185] (MonitorAgent_1): Tproc: 0.001522
NS shut down.
[2021-02-05 19:13:51.932632] (SineGeneratorAgent_1): Pack time: 0.001127
[2021-02-05 19:13:51.935690] (SineGeneratorAgent_1): Sending: [0.30901699]
[2021-02-05 19:13:51.935544] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.30901699]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:13:51.944128] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699])}
[2021-02-05 19:13:51.944378] (MonitorAgent_1): Tproc: 0.008396
[2021-02-05 19:13:52.930087] (SineGeneratorAgent_1): Pack time: 0.000108
[2021-02-05 19:13:52.930343] (SineGeneratorAgent_1): Sending: [0.36812455]
[2021-02-05 19:13:52.930548] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.36812455]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
```

(continues on next page)

(continued from previous page)

```
[2021-02-05 19:13:52.930905] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455])}
[2021-02-05 19:13:52.931004] (MonitorAgent_1): Tproc: 0.000381
[2021-02-05 19:13:53.930135] (SineGeneratorAgent_1): Pack time: 0.000144
[2021-02-05 19:13:53.930526] (SineGeneratorAgent_1): Sending: [0.42577929]
[2021-02-05 19:13:53.930537] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.42577929]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:13:53.930940] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455, 0.42577929])}
[2021-02-05 19:13:53.930993] (MonitorAgent_1): Tproc: 0.00038
[2021-02-05 19:13:54.932072] (SineGeneratorAgent_1): Pack time: 0.0009
[2021-02-05 19:13:54.934391] (SineGeneratorAgent_1): Sending: [0.48175367]
[2021-02-05 19:13:54.934275] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.48175367]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:13:54.936874] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455, 0.42577929, 0.48175367])}
[2021-02-05 19:13:54.937315] (MonitorAgent_1): Tproc: 0.002635
[2021-02-05 19:13:55.931558] (SineGeneratorAgent_1): Pack time: 0.000454
[2021-02-05 19:13:55.933193] (SineGeneratorAgent_1): Sending: [0.53582679]
[2021-02-05 19:13:55.933792] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.53582679]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:13:55.936039] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455, 0.42577929, 0.48175367, 0.53582679])}
[2021-02-05 19:13:55.936606] (MonitorAgent_1): Tproc: 0.002345
```

CHAPTER 17

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.

[1]:
```
# %load tutorial_2_math_agent.py
from agentMET4FOF.agents import (
    AgentMET4FOF,
    AgentNetwork,
    MonitorAgent,
    SineGeneratorAgent,
)


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

    # Define simple math functions.
    @staticmethod
    def divide_by_two(numerator: float) -> float:
```

(continues on next page)

(continued from previous page)

```
        return numerator / 2


class MultiMathAgent(AgentMET4FOF):

    _minus_param: float
    _plus_param: float

    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 = self.minus(message["data"], self._minus_param)
        plus_data = self.plus(message["data"], self._plus_param)

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

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

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


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
NS running on 127.0.0.1:3333 (127.0.0.1)
URI = PYRO:Pyro.NameServer@127.0.0.1:3333
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------------
|                                                          |
| Your agent network is starting up. Open your browser and |
| visit the agentMET4FOF dashboard on http://127.0.0.1:8050/ |
|                                                          |
------------------------------------------------------------

INFO [2021-02-05 19:18:07.585019] (SineGeneratorAgent_1): INITIALIZED
INFO [2021-02-05 19:18:07.619684] (MathAgent_1): INITIALIZED
INFO [2021-02-05 19:18:07.654192] (MultiMathAgent_1): INITIALIZED
INFO [2021-02-05 19:18:07.691566] (MonitorAgent_1): INITIALIZED
[2021-02-05 19:18:07.706815] (SineGeneratorAgent_1): Connected output module:␣
↪MathAgent_1
[2021-02-05 19:18:07.714664] (SineGeneratorAgent_1): Connected output module:␣
↪MultiMathAgent_1
[2021-02-05 19:18:07.724529] (SineGeneratorAgent_1): Connected output module:␣
↪MonitorAgent_1
[2021-02-05 19:18:07.738462] (MathAgent_1): Connected output module: MonitorAgent_1
[2021-02-05 19:18:07.750583] (MultiMathAgent_1): Connected output module:␣
↪MonitorAgent_1
SET STATE:   Running
[2021-02-05 19:18:08.589720] (SineGeneratorAgent_1): Pack time: 0.000765
[2021-02-05 19:18:08.592546] (MathAgent_1): Received: {'from': 'SineGeneratorAgent_1',
↪ 'data': array([0.]), 'senderType': 'SineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:18:08.595246] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.]), 'senderType': 'SineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:18:08.596230] (MultiMathAgent_1): Received: {'from':
↪'SineGeneratorAgent_1', 'data': array([0.]), 'senderType': 'SineGeneratorAgent',
↪'channel': 'default'}
[2021-02-05 19:18:08.595257] (SineGeneratorAgent_1): Sending: [0.]
[2021-02-05 19:18:08.593186] (MathAgent_1): Pack time: 0.000388
[2021-02-05 19:18:08.600139] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.])}
[2021-02-05 19:18:08.596854] (MultiMathAgent_1): Pack time: 0.000383
[2021-02-05 19:18:08.597959] (MultiMathAgent_1): Sending: {'minus': array([-0.5]),
↪'plus': array([0.5])}
[2021-02-05 19:18:08.593788] (MathAgent_1): Sending: [0.]
[2021-02-05 19:18:08.600521] (MonitorAgent_1): Tproc: 0.000931
[2021-02-05 19:18:08.598126] (MultiMathAgent_1): Tproc: 0.001735
[2021-02-05 19:18:08.594015] (MathAgent_1): Tproc: 0.001276
[2021-02-05 19:18:08.606396] (MonitorAgent_1): Received: {'from': 'MathAgent_1', 'data
↪': array([0.]), 'senderType': 'MathAgent', 'channel': 'default'}
[2021-02-05 19:18:08.607428] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.]), 'MathAgent_1': array([0.])}
[2021-02-05 19:18:08.607807] (MonitorAgent_1): Tproc: 0.001189
[2021-02-05 19:18:08.613784] (MonitorAgent_1): Received: {'from': 'MultiMathAgent_1',
↪'data': {'minus': array([-0.5]), 'plus': array([0.5])}, 'senderType':
↪'MultiMathAgent', 'channel': 'default'}
[2021-02-05 19:18:08.620942] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.]), 'MathAgent_1': array([0.]), 'MultiMathAgent_1': {'minus': array([-0.
↪5]), 'plus': array([0.5])}}
[2021-02-05 19:18:08.625194] (MonitorAgent_1): Tproc: 0.01115
[2021-02-05 19:18:09.590078] (SineGeneratorAgent_1): Pack time: 0.000561
[2021-02-05 19:18:09.592887] (SineGeneratorAgent_1): Sending: [0.06279052]
[2021-02-05 19:18:09.593616] (MathAgent_1): Received: {'from': 'SineGeneratorAgent_1',
↪ 'data': array([0.06279052]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
```

(continues on next page)

(continued from previous page)

```
[2021-02-05 19:18:09.593134] (MultiMathAgent_1): Received: {'from':
↪'SineGeneratorAgent_1', 'data': array([0.06279052]), 'senderType':
↪'SineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:18:09.598873] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.06279052]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:18:09.608525] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052]), 'MathAgent_1': array([0.]), 'MultiMathAgent_1': {
↪'minus': array([-0.5]), 'plus': array([0.5])}}
[2021-02-05 19:18:09.594706] (MathAgent_1): Pack time: 0.000551
[2021-02-05 19:18:09.596935] (MultiMathAgent_1): Pack time: 0.001516
[2021-02-05 19:18:09.613332] (MonitorAgent_1): Tproc: 0.013659
[2021-02-05 19:18:09.596653] (MathAgent_1): Sending: [0.03139526]
[2021-02-05 19:18:09.613588] (MultiMathAgent_1): Sending: {'minus': array([-0.
↪43720948]), 'plus': array([0.56279052])}
[2021-02-05 19:18:09.620995] (MonitorAgent_1): Received: {'from': 'MathAgent_1', 'data
↪': array([0.03139526]), 'senderType': 'MathAgent', 'channel': 'default'}
[2021-02-05 19:18:09.597111] (MathAgent_1): Tproc: 0.003054
[2021-02-05 19:18:09.616401] (MultiMathAgent_1): Tproc: 0.022068
[2021-02-05 19:18:09.631221] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052]), 'MathAgent_1': array([0.        , 0.03139526]),
↪'MultiMathAgent_1': {'minus': array([-0.5]), 'plus': array([0.5])}}
[2021-02-05 19:18:09.632022] (MonitorAgent_1): Tproc: 0.0105
[2021-02-05 19:18:09.637249] (MonitorAgent_1): Received: {'from': 'MultiMathAgent_1',
↪'data': {'minus': array([-0.43720948]), 'plus': array([0.56279052])}, 'senderType':
↪'MultiMathAgent', 'channel': 'default'}
[2021-02-05 19:18:09.651789] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052]), 'MathAgent_1': array([0.        , 0.03139526]),
↪'MultiMathAgent_1': {'minus': array([-0.5       , -0.43720948]), 'plus': array([0.5␣
↪      , 0.56279052])}}
[2021-02-05 19:18:09.652908] (MonitorAgent_1): Tproc: 0.014285
```

CHAPTER 18

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.

[2]:
```
# %load tutorial_3_multi_channel.py
from agentMET4FOF.agents import AgentMET4FOF, AgentNetwork, MonitorAgent
from agentMET4FOF.streams import SineGenerator, CosineGenerator


class MultiGeneratorAgent(AgentMET4FOF):

    _sine_stream: SineGenerator
    _cos_stream: CosineGenerator

    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
            self.send_output(sine_data["quantities"], channel="sine")
            self.send_output(cosine_data["quantities"], channel="cosine")
```

(continues on next page)

(continued from previous page)

```
class MultiOutputMathAgent(AgentMET4FOF):

    _minus_param: float
    _plus_param: float

    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 = self.minus(message["data"], self._minus_param)
            self.send_output({"cosine_minus": minus_data})
        elif message["channel"] == "sine":
            plus_data = self.plus(message["data"], self._plus_param)
            self.send_output({"sine_plus": plus_data})

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

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


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
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------------
|                                                          |
| Your agent network is starting up. Open your browser and |
| visit the agentMET4FOF dashboard on http://127.0.0.1:8050/ |
|                                                          |
------------------------------------------------------------

INFO [2021-02-05 19:22:24.987594] (MultiGeneratorAgent_1): INITIALIZED
INFO [2021-02-05 19:22:25.023399] (MultiOutputMathAgent_1): INITIALIZED
INFO [2021-02-05 19:22:25.051788] (MonitorAgent_1): INITIALIZED
[2021-02-05 19:22:25.071079] (MultiGeneratorAgent_1): Connected output module:␣
↪MultiOutputMathAgent_1
[2021-02-05 19:22:25.080910] (MultiGeneratorAgent_1): Connected output module:␣
↪MonitorAgent_1
[2021-02-05 19:22:25.098071] (MultiOutputMathAgent_1): Connected output module:␣
↪MonitorAgent_1
SET STATE:   Running
[2021-02-05 19:22:25.991167] (MultiGeneratorAgent_1): Pack time: 0.00034
[2021-02-05 19:22:25.991887] (MultiGeneratorAgent_1): Sending: [0.]
[2021-02-05 19:22:25.992459] (MonitorAgent_1): Received: {'from':
↪'MultiGeneratorAgent_1', 'data': array([0.]), 'senderType': 'MultiGeneratorAgent',
↪'channel': 'sine'}
[2021-02-05 19:22:25.995271] (MultiOutputMathAgent_1): Received: {'from':
↪'MultiGeneratorAgent_1', 'data': array([0.]), 'senderType': 'MultiGeneratorAgent',
↪'channel': 'sine'}
[2021-02-05 19:22:25.992155] (MultiGeneratorAgent_1): Pack time: 0.000109
[2021-02-05 19:22:25.992620] (MonitorAgent_1): Tproc: 1.4e-05
[2021-02-05 19:22:25.995647] (MultiOutputMathAgent_1): Pack time: 0.000224
[2021-02-05 19:22:25.992811] (MultiGeneratorAgent_1): Sending: [0.06279052]
[2021-02-05 19:22:25.993519] (MonitorAgent_1): Received: {'from':
↪'MultiGeneratorAgent_1', 'data': array([0.06279052]), 'senderType':
↪'MultiGeneratorAgent', 'channel': 'cosine'}
[2021-02-05 19:22:25.996258] (MultiOutputMathAgent_1): Sending: {'sine_plus':␣
↪array([0.5])}
[2021-02-05 19:22:25.996382] (MultiOutputMathAgent_1): Tproc: 0.000987
[2021-02-05 19:22:25.993658] (MonitorAgent_1): Tproc: 6e-06
[2021-02-05 19:22:25.996932] (MultiOutputMathAgent_1): Received: {'from':
↪'MultiGeneratorAgent_1', 'data': array([0.06279052]), 'senderType':
↪'MultiGeneratorAgent', 'channel': 'cosine'}
[2021-02-05 19:22:25.996166] (MonitorAgent_1): Received: {'from':
↪'MultiOutputMathAgent_1', 'data': {'sine_plus': array([0.5])}, 'senderType':
↪'MultiOutputMathAgent', 'channel': 'default'}
[2021-02-05 19:22:25.997141] (MultiOutputMathAgent_1): Pack time: 0.000105
[2021-02-05 19:22:25.996551] (MonitorAgent_1): Buffer: {'MultiOutputMathAgent_1': {
↪'sine_plus': array([0.5])}}
[2021-02-05 19:22:25.997460] (MultiOutputMathAgent_1): Sending: {'cosine_minus':␣
↪array([-0.43720948])}
[2021-02-05 19:22:25.996646] (MonitorAgent_1): Tproc: 0.00037
[2021-02-05 19:22:25.997542] (MultiOutputMathAgent_1): Tproc: 0.00053
[2021-02-05 19:22:25.997715] (MonitorAgent_1): Received: {'from':
↪'MultiOutputMathAgent_1', 'data': {'cosine_minus': array([-0.43720948])},
↪'senderType': 'MultiOutputMathAgent', 'channel': 'default'}
[2021-02-05 19:22:25.998228] (MonitorAgent_1): Buffer: {'MultiOutputMathAgent_1': {
↪'sine_plus': array([0.5]), 'cosine_minus': array([-0.43720948])}}
[2021-02-05 19:22:25.998299] (MonitorAgent_1): Tproc: 0.000488
[2021-02-05 19:22:26.992409] (MultiGeneratorAgent_1): Pack time: 0.000556
[2021-02-05 19:22:26.995214] (MultiOutputMathAgent_1): Received: {'from':
↪'MultiGeneratorAgent_1', 'data': array([0.12533323]), 'senderType':
↪'MultiGeneratorAgent', 'channel': 'sine'}
```

(continues on next page)

(continued from previous page)

```
[2021-02-05 19:22:26.995303] (MonitorAgent_1): Received: {'from':
↪'MultiGeneratorAgent_1', 'data': array([0.12533323]), 'senderType':
↪'MultiGeneratorAgent', 'channel': 'sine'}
[2021-02-05 19:22:26.994201] (MultiGeneratorAgent_1): Sending: [0.12533323]
[2021-02-05 19:22:26.996409] (MultiOutputMathAgent_1): Pack time: 0.000621
[2021-02-05 19:22:26.995688] (MonitorAgent_1): Tproc: 1.6e-05
[2021-02-05 19:22:26.995315] (MultiGeneratorAgent_1): Pack time: 0.000649
[2021-02-05 19:22:26.998020] (MultiOutputMathAgent_1): Sending: {'sine_plus':␣
↪array([0.62533323])}
[2021-02-05 19:22:26.999748] (MonitorAgent_1): Received: {'from':
↪'MultiGeneratorAgent_1', 'data': array([0.18738131]), 'senderType':
↪'MultiGeneratorAgent', 'channel': 'cosine'}
[2021-02-05 19:22:26.996890] (MultiGeneratorAgent_1): Sending: [0.18738131]
[2021-02-05 19:22:26.999032] (MultiOutputMathAgent_1): Tproc: 0.00333
[2021-02-05 19:22:27.000547] (MonitorAgent_1): Tproc: 1.8e-05
[2021-02-05 19:22:27.003974] (MonitorAgent_1): Received: {'from':
↪'MultiOutputMathAgent_1', 'data': {'sine_plus': array([0.62533323])}, 'senderType':
↪'MultiOutputMathAgent', 'channel': 'default'}
[2021-02-05 19:22:27.002023] (MultiOutputMathAgent_1): Received: {'from':
↪'MultiGeneratorAgent_1', 'data': array([0.18738131]), 'senderType':
↪'MultiGeneratorAgent', 'channel': 'cosine'}
[2021-02-05 19:22:27.009822] (MonitorAgent_1): Buffer: {'MultiOutputMathAgent_1': {
↪'sine_plus': array([0.5       , 0.62533323]), 'cosine_minus': array([-0.43720948])}}
[2021-02-05 19:22:27.003907] (MultiOutputMathAgent_1): Pack time: 0.001006
[2021-02-05 19:22:27.011142] (MonitorAgent_1): Tproc: 0.006396
[2021-02-05 19:22:27.006037] (MultiOutputMathAgent_1): Sending: {'cosine_minus':␣
↪array([-0.31261869])}
[2021-02-05 19:22:27.014405] (MonitorAgent_1): Received: {'from':
↪'MultiOutputMathAgent_1', 'data': {'cosine_minus': array([-0.31261869])},
↪'senderType': 'MultiOutputMathAgent', 'channel': 'default'}
[2021-02-05 19:22:27.007324] (MultiOutputMathAgent_1): Tproc: 0.004642
[2021-02-05 19:22:27.018020] (MonitorAgent_1): Buffer: {'MultiOutputMathAgent_1': {
↪'sine_plus': array([0.5       , 0.62533323]), 'cosine_minus': array([-0.43720948, -
↪0.31261869])}}
[2021-02-05 19:22:27.019178] (MonitorAgent_1): Tproc: 0.004049
```

CHAPTER 19

Tutorial 4 - A metrological datastream

In this tutorial we introduce the new metrologically enabled agents. We initialize an agent, which generates an infinite sine signal. The signal is generated from the built-in class `MetrologicalSineGenerator` which delivers on each call one timestamp and one value each with associated uncertainties.

The *MetrologicalSineGeneratorAgent* is based on the new class *agentMET4FOF.metrological_agents.MetrologicalAgent*. We only adapt the methods `init_parameters()` and `agent_loop()`. This we need to hand over an instance of the signal generating class and to generate the actual samples. The rest of the buffering and plotting logic is encapsulated inside of the new base classes.

[2]:
```
# %load tutorial_4_metrological_streams.py
from agentMET4FOF.agents import AgentNetwork
from agentMET4FOF.metrological_agents import MetrologicalAgent,␣
↪MetrologicalMonitorAgent
from agentMET4FOF.metrological_streams import (
    MetrologicalDataStreamMET4FOF,
    MetrologicalSineGenerator,
)


class MetrologicalSineGeneratorAgent(MetrologicalAgent):
    """An agent streaming a sine signal

    Takes samples from an instance of :py:class:`MetrologicalSineGenerator` and pushes
    them sample by sample to connected agents via its output channel.
    """

    # The datatype of the stream will be MetrologicalSineGenerator.
    _stream: MetrologicalDataStreamMET4FOF

    def init_parameters(
        self,
        signal: MetrologicalDataStreamMET4FOF = MetrologicalSineGenerator(),
        **kwargs
    ):
```

(continues on next page)

(continued from previous page)

```
        """Initialize the input data stream

        Parameters
        ----------
        signal : MetrologicalDataStreamMET4FOF
            the underlying signal for the generator
        """
        self._stream = signal
        super().init_parameters()
        self.set_output_data(channel="default", metadata=self._stream.metadata)

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

        On state *Running* the agent will extract sample by sample the input
        datastream's content and push it into its output buffer.
        """
        if self.current_state == "Running":
            self.set_output_data(channel="default", data=[self._stream.next_sample()])
            super().agent_loop()


def demonstrate_metrological_stream():

    # start agent network server
    agent_network = AgentNetwork(dashboard_modules=True)

    # Initialize signal generating class outside of agent framework.
    signal = MetrologicalSineGenerator()

    # Initialize metrologically enabled agent taking name from signal source metadata.
    source_name = signal.metadata.metadata["device_id"]
    source_agent = agent_network.add_agent(
        name=source_name, agentType=MetrologicalSineGeneratorAgent
    )
    source_agent.init_parameters(signal)

    # Initialize metrologically enabled plotting agent.
    monitor_agent = agent_network.add_agent(
        "MonitorAgent",
        agentType=MetrologicalMonitorAgent,
        buffer_size=50,
    )

    # Bind agents.
    source_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_metrological_stream()
```

```
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

------------------------------------------------------------
|                                                          |
| Your agent network is starting up. Open your browser and |
| visit the agentMET4FOF dashboard on http://127.0.0.1:8050/ |
|                                                          |
------------------------------------------------------------

INFO [2021-02-05 19:24:37.900901] (SineGenerator): INITIALIZED
INFO [2021-02-05 19:24:37.939928] (MonitorAgent): INITIALIZED
[2021-02-05 19:24:37.955210] (SineGenerator): Connected output module: MonitorAgent
SET STATE:   Running
[2021-02-05 19:24:38.907664] (SineGenerator): Pack time: 0.000336
[2021-02-05 19:24:38.909019] (SineGenerator): Sending: [array([[0.        , 0.
↪, 0.68494649, 0.25      ]]), <time_series_metadata.scheme.MetaData object at␣
↪0x7f424e871730>]
[2021-02-05 19:24:38.910202] (MonitorAgent): Received: {'from': 'SineGenerator', 'data
↪': array([[0.        , 0.        , 0.68494649, 0.25      ]]), 'metadata': <time_
↪series_metadata.scheme.MetaData object at 0x7f424e822bb0>, 'senderType':
↪'MetrologicalSineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:24:38.911237] (MonitorAgent): Buffer: {'SineGenerator': {'data':␣
↪array([[0.        , 0.        , 0.68494649, 0.25      ]]), 'metadata': [<time_
↪series_metadata.scheme.MetaData object at 0x7f424e822bb0>]}}
[2021-02-05 19:24:38.911459] (MonitorAgent): Tproc: 0.000968
[2021-02-05 19:24:39.907438] (SineGenerator): Pack time: 0.000403
[2021-02-05 19:24:39.909067] (MonitorAgent): Received: {'from': 'SineGenerator', 'data
↪': array([[0.002     , 0.        , 0.42028269, 0.25      ]]), 'metadata': <time_
↪series_metadata.scheme.MetaData object at 0x7f424e822c70>, 'senderType':
↪'MetrologicalSineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:24:39.908668] (SineGenerator): Sending: [array([[0.002     , 0.
↪, 0.42028269, 0.25      ]]), <time_series_metadata.scheme.MetaData object at␣
↪0x7f424e871730>]
[2021-02-05 19:24:39.910652] (MonitorAgent): Buffer: {'SineGenerator': {'data':␣
↪array([[0.        , 0.        , 0.68494649, 0.25      ],
       [0.002     , 0.        , 0.42028269, 0.25      ]]), 'metadata': [<time_series_
↪metadata.scheme.MetaData object at 0x7f424e822bb0>, <time_series_metadata.scheme.
↪MetaData object at 0x7f424e822c70>]}}
[2021-02-05 19:24:39.911060] (MonitorAgent): Tproc: 0.001731
[2021-02-05 19:24:40.908816] (SineGenerator): Pack time: 0.000738
[2021-02-05 19:24:40.912960] (SineGenerator): Sending: [array([[0.004     , 0.
↪, 1.54415656, 0.25      ]]), <time_series_metadata.scheme.MetaData object at␣
↪0x7f424e871730>]
[2021-02-05 19:24:40.913508] (MonitorAgent): Received: {'from': 'SineGenerator', 'data
↪': array([[0.004     , 0.        , 1.54415656, 0.25      ]]), 'metadata': <time_
↪series_metadata.scheme.MetaData object at 0x7f424e822ca0>, 'senderType':
↪'MetrologicalSineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:24:40.917793] (MonitorAgent): Buffer: {'SineGenerator': {'data':␣
↪array([[0.        , 0.        , 0.68494649, 0.25      ],
       [0.002     , 0.        , 0.42028269, 0.25      ],
       [0.004     , 0.        , 1.54415656, 0.25      ]]), 'metadata': [<time_series_
↪metadata.scheme.MetaData object at 0x7f424e822bb0>, <time_series_metadata.scheme.
↪MetaData object at 0x7f424e822c70>, <time_series_metadata.scheme.MetaData object at␣
↪0x7f424e822ca0>]}}
```

(continues on next page)

(continued from previous page)

```
[2021-02-05 19:24:40.919608] (MonitorAgent): Tproc: 0.005357
[2021-02-05 19:24:41.907632] (SineGenerator): Pack time: 0.000437
[2021-02-05 19:24:41.909056] (SineGenerator): Sending: [array([[0.006    , 0.       ,␣
↪0.9256774, 0.25     ]]), <time_series_metadata.scheme.MetaData object at␣
↪0x7f424e871730>]
[2021-02-05 19:24:41.909673] (MonitorAgent): Received: {'from': 'SineGenerator', 'data
↪': array([[0.006    , 0.       , 0.9256774, 0.25     ]]), 'metadata': <time_series_
↪metadata.scheme.MetaData object at 0x7f424e822d00>, 'senderType':
↪'MetrologicalSineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:24:41.913785] (MonitorAgent): Buffer: {'SineGenerator': {'data':␣
↪array([[0.        , 0.        , 0.68494649, 0.25      ],
       [0.002     , 0.        , 0.42028269, 0.25      ],
       [0.004     , 0.        , 1.54415656, 0.25      ],
       [0.006     , 0.        , 0.9256774 , 0.25      ]]), 'metadata': [<time_series_
↪metadata.scheme.MetaData object at 0x7f424e822bb0>, <time_series_metadata.scheme.
↪MetaData object at 0x7f424e822c70>, <time_series_metadata.scheme.MetaData object at␣
↪0x7f424e822ca0>, <time_series_metadata.scheme.MetaData object at 0x7f424e822d00>]}}
[2021-02-05 19:24:41.914125] (MonitorAgent): Tproc: 0.004137
[2021-02-05 19:24:42.909007] (SineGenerator): Pack time: 0.000784
[2021-02-05 19:24:42.911438] (SineGenerator): Sending: [array([[0.008    , 0.       ␣
↪, 1.12170929, 0.25      ]]), <time_series_metadata.scheme.MetaData object at␣
↪0x7f424e871730>]
[2021-02-05 19:24:42.913822] (MonitorAgent): Received: {'from': 'SineGenerator', 'data
↪': array([[0.008    , 0.       , 1.12170929, 0.25      ]]), 'metadata': <time_
↪series_metadata.scheme.MetaData object at 0x7f424e822490>, 'senderType':
↪'MetrologicalSineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:24:42.920323] (MonitorAgent): Buffer: {'SineGenerator': {'data':␣
↪array([[0.        , 0.        , 0.68494649, 0.25      ],
       [0.002     , 0.        , 0.42028269, 0.25      ],
       [0.004     , 0.        , 1.54415656, 0.25      ],
       [0.006     , 0.        , 0.9256774 , 0.25      ],
       [0.008     , 0.        , 1.12170929, 0.25      ]]), 'metadata': [<time_series_
↪metadata.scheme.MetaData object at 0x7f424e822bb0>, <time_series_metadata.scheme.
↪MetaData object at 0x7f424e822c70>, <time_series_metadata.scheme.MetaData object at␣
↪0x7f424e822ca0>, <time_series_metadata.scheme.MetaData object at 0x7f424e822d00>,
↪<time_series_metadata.scheme.MetaData object at 0x7f424e822490>]}}
[2021-02-05 19:24:42.921078] (MonitorAgent): Tproc: 0.006613
```

CHAPTER 20

Tutorial 5 - Building coalitions

We demonstrate the use of Coalition of agents to group agents together. Rationale of grouping depends on the users and application. For example, we can group sensors which are measuring the same measurand. To this end, the coalition consists of a list of agent names and provides aesthetic differences in the dashboard.

All coalitions visible by the `agent_network` can be accessed via `agent_network.coalitions`.

[1]:
```
# %load tutorial_5_coalition.py
from agentMET4FOF.agents import AgentNetwork, MonitorAgent, SineGeneratorAgent


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

    # Initialize agents by adding them to the agent network.
    gen_agent_1 = agent_network.add_agent(agentType=SineGeneratorAgent)
    gen_agent_2 = agent_network.add_agent(agentType=SineGeneratorAgent)
    monitor_agent = agent_network.add_agent(agentType=MonitorAgent)

    # bind generator agents outputs to monitor
    agent_network.bind_agents(gen_agent_1, monitor_agent)
    agent_network.bind_agents(gen_agent_2, monitor_agent)

    # setup health coalition group
    agent_network.add_coalition(
        "REDUNDANT_SENSORS", [gen_agent_1, gen_agent_2, monitor_agent]
    )

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

    # Allow for shutting down the network after execution
    return agent_network
```

(continues on next page)

(continued from previous page)

```
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

------------------------------------------------------------
|                                                          |
| Your agent network is starting up. Open your browser and |
| visit the agentMET4FOF dashboard on http://127.0.0.1:8050/ |
|                                                          |
------------------------------------------------------------

INFO [2021-02-05 19:34:50.491737] (SineGeneratorAgent_1): INITIALIZED
INFO [2021-02-05 19:34:50.524928] (SineGeneratorAgent_2): INITIALIZED
INFO [2021-02-05 19:34:50.557538] (MonitorAgent_1): INITIALIZED
[2021-02-05 19:34:50.574726] (SineGeneratorAgent_1): Connected output module:␣
↪MonitorAgent_1
[2021-02-05 19:34:50.583164] (SineGeneratorAgent_2): Connected output module:␣
↪MonitorAgent_1
SET STATE:   Running
[2021-02-05 19:34:51.499491] (SineGeneratorAgent_1): Pack time: 0.001512
[2021-02-05 19:34:51.502797] (SineGeneratorAgent_1): Sending: [0.]
[2021-02-05 19:34:51.507439] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.]), 'senderType': 'SineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:34:51.543076] (SineGeneratorAgent_2): Pack time: 0.002763
[2021-02-05 19:34:51.509952] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.])}
[2021-02-05 19:34:51.546875] (SineGeneratorAgent_2): Sending: [0.]
[2021-02-05 19:34:51.511171] (MonitorAgent_1): Tproc: 0.002834
[2021-02-05 19:34:51.544997] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪2', 'data': array([0.]), 'senderType': 'SineGeneratorAgent', 'channel': 'default'}
[2021-02-05 19:34:51.554497] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.]), 'SineGeneratorAgent_2': array([0.])}
[2021-02-05 19:34:51.555370] (MonitorAgent_1): Tproc: 0.009853
[2021-02-05 19:34:52.495894] (SineGeneratorAgent_1): Pack time: 0.00018
[2021-02-05 19:34:52.496930] (SineGeneratorAgent_1): Sending: [0.06279052]
[2021-02-05 19:34:52.497336] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.06279052]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:52.498583] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052]), 'SineGeneratorAgent_2': array([0.])}
[2021-02-05 19:34:52.498796] (MonitorAgent_1): Tproc: 0.001265
[2021-02-05 19:34:52.528407] (SineGeneratorAgent_2): Pack time: 0.000181
[2021-02-05 19:34:52.529454] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪2', 'data': array([0.06279052]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:52.529242] (SineGeneratorAgent_2): Sending: [0.06279052]
[2021-02-05 19:34:52.530661] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052]), 'SineGeneratorAgent_2': array([0.        , 0.
↪06279052])}
[2021-02-05 19:34:52.530827] (MonitorAgent_1): Tproc: 0.001183
[2021-02-05 19:34:53.496614] (SineGeneratorAgent_1): Pack time: 0.000331
```

(continues on next page)

(continued from previous page)

```
[2021-02-05 19:34:53.497841] (SineGeneratorAgent_1): Sending: [0.12533323]
[2021-02-05 19:34:53.499000] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.12533323]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:53.535469] (SineGeneratorAgent_2): Pack time: 0.000494
[2021-02-05 19:34:53.503180] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323]), 'SineGeneratorAgent_2': array([0.      ␣
↪  , 0.06279052])}
[2021-02-05 19:34:53.537256] (SineGeneratorAgent_2): Sending: [0.12533323]
[2021-02-05 19:34:53.503676] (MonitorAgent_1): Tproc: 0.004153
[2021-02-05 19:34:53.541731] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪2', 'data': array([0.12533323]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:53.545969] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323]), 'SineGeneratorAgent_2': array([0.      ␣
↪  , 0.06279052, 0.12533323])}
[2021-02-05 19:34:53.554543] (MonitorAgent_1): Tproc: 0.011906
NS shut down.
[2021-02-05 19:34:54.497690] (SineGeneratorAgent_1): Pack time: 0.000662
[2021-02-05 19:34:54.504170] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.18738131]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:54.502583] (SineGeneratorAgent_1): Sending: [0.18738131]
[2021-02-05 19:34:54.515421] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131]), 'SineGeneratorAgent_2':␣
↪array([0.        , 0.06279052, 0.12533323])}
[2021-02-05 19:34:54.528529] (SineGeneratorAgent_2): Pack time: 0.000237
[2021-02-05 19:34:54.529196] (SineGeneratorAgent_2): Sending: [0.18738131]
[2021-02-05 19:34:54.519541] (MonitorAgent_1): Tproc: 0.014505
[2021-02-05 19:34:54.531254] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪2', 'data': array([0.18738131]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:54.532868] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131]), 'SineGeneratorAgent_2':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131])}
[2021-02-05 19:34:54.533451] (MonitorAgent_1): Tproc: 0.001834
[2021-02-05 19:34:55.497332] (SineGeneratorAgent_1): Pack time: 0.00057
[2021-02-05 19:34:55.499010] (SineGeneratorAgent_1): Sending: [0.24868989]
[2021-02-05 19:34:55.500133] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.24868989]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:55.533920] (SineGeneratorAgent_2): Pack time: 0.001052
[2021-02-05 19:34:55.538313] (SineGeneratorAgent_2): Sending: [0.24868989]
[2021-02-05 19:34:55.503852] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989]),
↪'SineGeneratorAgent_2': array([0.        , 0.06279052, 0.12533323, 0.18738131])}
[2021-02-05 19:34:55.504382] (MonitorAgent_1): Tproc: 0.003753
[2021-02-05 19:34:55.538319] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪2', 'data': array([0.24868989]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:55.544134] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989]),
↪'SineGeneratorAgent_2': array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.
↪24868989])}
[2021-02-05 19:34:55.545007] (MonitorAgent_1): Tproc: 0.005636
[2021-02-05 19:34:56.496439] (SineGeneratorAgent_1): Pack time: 0.000345
[2021-02-05 19:34:56.499405] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.30901699]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
```

(continues on next page)

(continued from previous page)

```
[2021-02-05 19:34:56.497380] (SineGeneratorAgent_1): Sending: [0.30901699]
[2021-02-05 19:34:56.503408] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699]), 'SineGeneratorAgent_2': array([0.        , 0.06279052, 0.
↪12533323, 0.18738131, 0.24868989])}
[2021-02-05 19:34:56.530998] (SineGeneratorAgent_2): Pack time: 0.000148
[2021-02-05 19:34:56.503616] (MonitorAgent_1): Tproc: 0.003954
[2021-02-05 19:34:56.531354] (SineGeneratorAgent_2): Sending: [0.30901699]
[2021-02-05 19:34:56.532137] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪2', 'data': array([0.30901699]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:56.532882] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699]), 'SineGeneratorAgent_2': array([0.        , 0.06279052, 0.
↪12533323, 0.18738131, 0.24868989,
       0.30901699])}
[2021-02-05 19:34:56.533007] (MonitorAgent_1): Tproc: 0.000751
[2021-02-05 19:34:57.497217] (SineGeneratorAgent_1): Pack time: 0.000607
[2021-02-05 19:34:57.498990] (SineGeneratorAgent_1): Sending: [0.36812455]
[2021-02-05 19:34:57.499886] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.36812455]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:57.529841] (SineGeneratorAgent_2): Pack time: 0.000718
[2021-02-05 19:34:57.505595] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455]), 'SineGeneratorAgent_2': array([0.        , 0.
↪06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699])}
[2021-02-05 19:34:57.531642] (SineGeneratorAgent_2): Sending: [0.36812455]
[2021-02-05 19:34:57.506484] (MonitorAgent_1): Tproc: 0.006025
[2021-02-05 19:34:57.548550] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪2', 'data': array([0.36812455]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:57.568741] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455]), 'SineGeneratorAgent_2': array([0.        , 0.
↪06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455])}
[2021-02-05 19:34:57.570422] (MonitorAgent_1): Tproc: 0.021339
[2021-02-05 19:34:58.495766] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪1', 'data': array([0.42577929]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:58.496176] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455, 0.42577929]), 'SineGeneratorAgent_2': array([0.
↪, 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455])}
[2021-02-05 19:34:58.495362] (SineGeneratorAgent_1): Pack time: 7.8e-05
[2021-02-05 19:34:58.496226] (MonitorAgent_1): Tproc: 0.000388
[2021-02-05 19:34:58.495629] (SineGeneratorAgent_1): Sending: [0.42577929]
[2021-02-05 19:34:58.527965] (SineGeneratorAgent_2): Pack time: 9.5e-05
[2021-02-05 19:34:58.528730] (MonitorAgent_1): Received: {'from': 'SineGeneratorAgent_
↪2', 'data': array([0.42577929]), 'senderType': 'SineGeneratorAgent', 'channel':
↪'default'}
[2021-02-05 19:34:58.528211] (SineGeneratorAgent_2): Sending: [0.42577929]
[2021-02-05 19:34:58.529372] (MonitorAgent_1): Buffer: {'SineGeneratorAgent_1':␣
↪array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
```

(continues on next page)

(continued from previous page)

```
       0.30901699, 0.36812455, 0.42577929]), 'SineGeneratorAgent_2': array([0.
↪, 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455, 0.42577929])}
[2021-02-05 19:34:58.529423] (MonitorAgent_1): Tproc: 0.000622
```

CHAPTER 21

Tutorial 6 - Using a different backend

By default, “osbrain” backend offers real connectivity between agents (each agent has its own port & IP address) in distributed systems (e,g connecting agents from raspberry pis to PCs, etc), which explains why it is harder to debug.

In the “mesa” backend, there’s only one real timer which is started in the AgentNetwork, and every timer tick will advance the agent actions by calling `step()` which includes `agent_loop` and `on_received_message`. Moreover, in the “mesa” backend, agents do not have their own port and IP addresses, they are simulated objects to emulate the behaviour of distributed agents. Hence, “osbrain” is closer to deployment phase, whereas mesa is suited for the simulation/designing phase. To switch between the backends, simply pass the backend parameter to either “mesa” or “osbrain” in the AgentNetwork instantiation.

[2]:
```
# %load tutorial_6_mesa_backend.py
from agentMET4FOF.agents import AgentNetwork, MonitorAgent, SineGeneratorAgent


def demonstrate_mesa_backend():

    # Start agent network and specify backend via the corresponding keyword parameter.
    _agent_network = AgentNetwork(backend="mesa")

    # Initialize agents by adding them to the agent network.
    sine_agent = _agent_network.add_agent(agentType=SineGeneratorAgent)
    monitor_agent = _agent_network.add_agent(agentType=MonitorAgent, buffer_size=200)
    sine_agent.bind_output(monitor_agent)

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

    return _agent_network


if __name__ == "__main__":
    demonstrate_mesa_backend()
```

```
------------------------------------------------------------
|                                                          |
| Your agent network is starting up. Open your browser and |
| visit the agentMET4FOF dashboard on http://127.0.0.1:8050/ |
|                                                          |
------------------------------------------------------------
[2021-02-05 19:39:41.143529] (SineGeneratorAgent_1): INITIALIZED

[2021-02-05 19:39:41.143892] (MonitorAgent_1): INITIALIZED
[2021-02-05 19:39:41.143941] (SineGeneratorAgent_1): Connected output module:␣
↪MonitorAgent_1
SET STATE:   Running
```

CHAPTER 22

agentMET4FOF agents

class agentMET4FOF.agents.**AgentBuffer**(*buffer_size: int = 1000*)

Buffer class which is instantiated in every agent to store data incrementally

This buffer is necessary to handle multiple inputs coming from agents.

We can access the buffer like a dict with exposed functions such as .values(), .keys() and .items(). The actual dict object is stored in the variable *buffer*.

buffer

The buffer can be a dict of iterables, or a dict of dict of iterables for nested named data. The keys are the names of agents.

Type dict of iterables or dict of dicts of iterables

buffer_size

The total number of elements to be stored in the agent *buffer*

supported_datatypes

List of all types supported and thus properly handled by the buffer. Defaults to `np.ndarray`, list and Pandas `DataFrame`

Type list of types

buffer_filled(*agent_from: Optional[str] = None*) → bool

Checks whether buffer is filled, by comparing against the *buffer_size*

For nested dict, this returns True if any of the iterables is beyond the *buffer_size*. For any of the dict values , which is not one of *supported_datatypes* this returns None.

Parameters **agent_from** (*str, optional*) – Name of input agent in the buffer dict to be looked up. If `agent_from` is not provided, we check for all iterables in the buffer (default).

Returns True if either the or any of the iterables has reached *buffer_size* or None in case none of the values is of one of the supported datatypes. False if all present iterable can take at least one more element according to *buffer_size*.

Return type bool or None

check_supported_datatype(*obj: object*) → bool
Checks whether *value* is an object of one of the supported data types

Parameters **obj** (*object*) – Value to be checked

Returns **result** – True if value is an object of one of the supported data types, False if not

Return type boolean

clear(*agent_from: Optional[str] = None*)
Clears the data in the buffer

Parameters **agent_from** (*str, optional*) – Name of agent, if `agent_from` is not given, the entire buffer is flushed. (default)

items()
Interface to access the internal dict's items()

keys()
Interface to access the internal dict's keys()

popleft(*n: Optional[int] = 1*) → Union[Dict[KT, VT], numpy.ndarray, list, pandas.core.frame.DataFrame]
Pops the first n entries in the buffer

Parameters **n** (*int*) – Number of elements to retrieve from buffer

Returns

- dict, `np.ndarray`, list or Pandas
- `DataFrame` – The retrieved elements

store(*agent_from: Union[Dict[str, Union[numpy.ndarray, list, pandas.core.frame.DataFrame]], str], data: Union[numpy.ndarray, list, pandas.core.frame.DataFrame, float, int] = None, concat_axis: Optional[int] = 0*)
Stores data into *buffer* 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 (keys `from` and `data` present'), otherwise we expect it to be name of agent sender and `data` will need to be passed as parameter
- **data** (*np.ndarray, DataFrame, list, float or int*) – Not used if `agent_from` is a dict. Otherwise `data` is compulsory.
- **concat_axis** (*int, optional*) – axis to concatenate on with the buffering for numpy arrays. Default is 0.

update(*agent_from: Union[Dict[str, Union[numpy.ndarray, list, pandas.core.frame.DataFrame]], str], data: Union[numpy.ndarray, list, pandas.core.frame.DataFrame, float, int] = None*)
Overrides data in the buffer dict keyed by `agent_from` with value `data`

If `data` is a single value, this converts it into a list first before storing in the buffer dict.

Parameters

- **agent_from** (*str*) – Name of agent sender
- **data** (*np.ndarray, DataFrame, list, float or int*) – New incoming data

values()
Interface to access the internal dict's values()

class agentMET4FOF.agents.**AgentMET4FOF**(*name=''*, *host=None*, *serializer=None*, *transport=None*, *attributes=None*, *backend='osbrain'*, *mesa_model=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

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

buffer_clear(*agent_name: Optional[str] = None*)
Empties buffer which is a dict indexed by the *agent_name*.

Parameters **agent_name** (*str, optional*) – Key of the memory dict, which can be the name of input agent, or self.name. If not supplied (default), we assume to clear the entire memory.

buffer_filled(*agent_name=None*)
Checks whether the internal buffer has been filled to the maximum allowed specified by self.buffer_size

Parameters **agent_name** (*str*) – Index of the buffer which is the name of input agent.

Returns **status of buffer filled**

Return type boolean

buffer_store(*agent_from: str*, *data=None*, *concat_axis=0*)
Updates data stored in *self.buffer* 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** (*str*) – Name of agent sender
- **data** – Any supported data which can be stored in dict as buffer. See AgentBuffer for more information.

get_attr(*attr*)
Return the specified attribute of the agent.

Parameters **name** – Name of the attribute to be retrieved.

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(*buffer_size=1000*, *log_mode=True*)
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

buffer_size
The total number of elements to be stored in the agent `buffer` When total elements exceeds this number, the latest elements will be replaced with the incoming data elements
Type int

init_agent_loop(*loop_wait: Optional[int] = None*)
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_buffer(*buffer_size*)
A method to initialise the buffer. By overriding this method, user can provide a custom buffer, instead of the regular AgentBuffer. This can be used, for example, to provide a MetrologicalAgentBuffer in the metrological agents.

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_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)

set_attr(***kwargs*)
Set object attributes.

Parameters **kwargs** (*[name, value]*) – Keyword arguments will be used to set the object attributes.

shutdown()
Cleanly stop and shut down the agent assuming the agent is running.

Will let the main thread do the tear down.

step()
Used for MESA backend only. Behaviour on every update step.

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

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*, *backend='osbrain'*, *mesa_update_interval=0.1*)
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*, *buffer_size=1000*, *ip_addr=None*, *loop_wait=None*, ***kwargs*)
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

add_coalition(*name='Coalition_1'*, *agents=[]*)
Instantiates a coalition of agents.

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*)
Only for osbrain backend. Connects to an existing AgentNetwork.

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

get_mode()

Returns **state** – State of Agent Network

Return type str

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()
Shuts down the entire agent network and all agents

start_server_mesa()
Starts a new AgentNetwork for Mesa and initializes `_controller`

Handles the initialisation for `backend == "mesa"`. Involves spawning two nested objects `mesa_model` and `_controller` and calls `start_mesa_timer()`.

start_server_osbrain(*ip_addr: str = '127.0.0.1'*, *port: int = 3333*)
Starts a new AgentNetwork for osBrain and initializes `_controller`

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*, *backend='osbrain'*, *mesa_model=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>*, *pretrain_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.**MesaModel**
A MESA Model

shutdown()
Shutdown entire MESA model with all agents and schedulers

step()
Advance the model by one step.

class agentMET4FOF.agents.**MonitorAgent**(*name=''*, *host=None*, *serializer=None*, *transport=None*, *attributes=None*, *backend='osbrain'*, *mesa_model=None*)
Unique Agent for storing plots and data from messages received from input agents.

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

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

custom_plot_function
a custom plot function that can be provided to handle the data in the monitor agents buffer (see *AgentMET4FOF* for details). The function gets provided with the content (value) of the buffer and with the string of the sender agent's name as stored in the buffer's keys. Additionally any other parameters can be provided as a dict in custom_plot_parameters.

Type callable

custom_plot_parameters
a custom dictionary of parameters that shall be provided to each call of the custom_plot_function

Type dict

init_parameters(*plot_filter: Optional[List[str]] = None, custom_plot_function: Optional[Callable[[...], plotly.graph_objs._scatter.Scatter]] = None, **kwargs*)
Initialize the monitor agent's parameters

Parameters

- **plot_filter** (*list of str, optional*) – 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
- **custom_plot_function** (*callable, optional*) – a custom plot function that can be provided to handle the data in the monitor agents buffer (see *AgentMET4FOF* for details). The function gets provided with the content (value) of the buffer and with the string of the sender agent's name as stored in the buffer's keys. Additionally any other parameters can be provided as a dict in custom_plot_parameters. By default the data gets plotted as shown in the various tutorials.
- **kwargs** (*Any*) – custom key word parameters that shall be provided to each call of the *custom_plot_function*

on_received_message(*message*)
Handles incoming data from 'default' and 'plot' channels.

Stores 'default' data into `buffer` and 'plot' data into *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: Dict[str, Any]*)
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.

class agentMET4FOF.agents.**SineGeneratorAgent**(*name=''*, *host=None*, *serializer=None*, *transport=None*, *attributes=None*, *backend='osbrain'*, *mesa_model=None*)
An agent streaming a sine signal

Takes samples from the SineGenerator and pushes them sample by sample to connected agents via its output channel.

agent_loop()
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 *AgentMET4FOF.send_output()*.

init_parameters(*sfreq=500*, *sine_freq=5*)
Initialize the input data

Initialize the input data stream as an instance of the SineGenerator class.

Parameters

- **sfreq** (*int*) – sampling frequency for the underlying signal
- **sine_freq** (*float*) – frequency of the generated sine wave

CHAPTER 23

agentMET4FOF streams

class agentMET4FOF.streams.**CosineGenerator**(*sfreq=500*, *cosine_freq=5*)

Built-in class of cosine wave generator which inherits all methods and attributes from *DataStreamMET4FOF*. *cosine_wave_function()* is a custom defined function which has a required keyword `time` as argument and any number of optional additional arguments (e.g `cosine_freq`) to be supplied to the *DataStreamMET4FOF.set_generator_function()*.

Parameters

- **sfreq** (*int*) – sampling frequency which determines the time step when *next_sample()* is called
- **F** (*int*) – frequency of wave function

cosine_wave_function(*time*, *cosine_freq=50*)

A simple cosine wave generator

class agentMET4FOF.streams.**DataStreamMET4FOF**

Abstract class for creating datastreams.

Data can be fetched sequentially using *next_sample()* or all at once *all_samples()*. This increments the internal sample index *_sample_idx*.

For sensors data, we assume:

- The format shape for 2D data stream (timesteps, n_sensors)
- The format shape for 3D data stream (num_cycles, timesteps , n_sensors)

To create a new DataStreamMET4FOF class, inherit this class and call *set_metadata()* in the constructor. Choose one of two types of datastreams to be created:

- from dataset file (*set_data_source()*), or
- a waveform generator function (*set_generator_function()*).

Alternatively, override the *next_sample()* function if neither option suits the application. For generator functions, *sfreq* is a required variable to be set on *init* which sets the sampling frequency and the time-step which occurs when *next_sample()* is called.

For an example implementation of using generator function, see the built-in *SineGenerator* class. See tutorials for more implementations.

_quantities
Measured quantities such as sensors readings

Type Union[List, DataFrame, np.ndarray]

_target
Target label in the context of machine learning. This can be Remaining Useful Life in predictive maintenance application. Note this can be an unobservable variable in real-time and applies only for validation during offline analysis.

Type Union[List, DataFrame, np.ndarray]

_time
`dtype` can be either `float` or `datetime64` to indicate the time when the *_quantities* were measured.

Type Union[List, DataFrame, np.ndarray]

_current_sample_quantities
Last returned measured quantities from a call to *next_sample()*

Type Union[List, DataFrame, np.ndarray]

_current_sample_target
Last returned target labels from a call to *next_sample()*

Type Union[List, DataFrame, np.ndarray]

_current_sample_time
`dtype` can be either `float` or `datetime64` to indicate the time when the *_current_sample_quantities* were measured.

Type Union[List, DataFrame, np.ndarray]

_sample_idx
Current sample index

Type int

_n_samples
Total number of samples

Type int

_data_source_type
Explicitly account for the data source type: either "function" or "dataset"

Type str

_generator_function
A generator function which takes in at least one argument `time` which will be used in *next_sample()*

Type Callable

_generator_parameters
Any additional keyword arguments to be supplied to the generator function. The generator function call for every sample will be supplied with the `**generator_parameters`.

Type Dict

sfreq
Sampling frequency

Type int

_metadata
The quantities metadata as time_series_metadata.scheme.MetaData

Type MetaData

_default_generator_function(*time*)
This is the default generator function used, if non was specified

Parameters **time** (*Union[List, DataFrame, np.ndarray]*) –

_next_sample_data_source(*batch_size: int = 1*) → Dict[str, Union[List[T], pandas.core.frame.DataFrame, numpy.ndarray]]
Internal method for fetching latest samples from a dataset.

Parameters **batch_size** (*int*) – number of batches to get from data stream

Returns **samples** – {'quantities':current_sample_quantities, 'target': current_sample_target}

Return type Dict

_next_sample_generator(*batch_size: int = 1*) → Dict[str, numpy.ndarray]
Internal method for generating a batch of samples from the generator function.

_set_data_source_type(*dt_type: str = 'function'*)
To explicitly account for the type of data source: either from dataset, or a generator function.

Parameters **dt_type** (*str*) – Either "function" or "dataset"

all_samples() → Dict[str, Union[List[T], pandas.core.frame.DataFrame, numpy.ndarray]]
Returns all the samples in the data stream

Returns **samples** – {'x': current_sample_x, 'y': current_sample_y}

Return type Dict

next_sample(*batch_size: int = 1*)
Fetches the latest batch_size samples from the iterables: quantities, time and target. This advances the internal pointer _sample_idx by batch_size.

Parameters **batch_size** (*int*) – number of batches to get from data stream

Returns **samples** – {'time':current_sample_time, 'quantities': current_sample_quantities, 'target':current_sample_target}

Return type Dict

randomize_data()
Randomizes the provided quantities, useful in machine learning contexts

set_data_source(*quantities: Union[List[T], pandas.core.frame.DataFrame, numpy.ndarray] = None, target: Union[List[T], pandas.core.frame.DataFrame, numpy.ndarray, None] = None, time: Union[List[T], pandas.core.frame.DataFrame, numpy.ndarray, None] = None*)
This sets the data source by providing up to three iterables: quantities , time and target which are assumed to be aligned.

For sensors data, we assume: The format shape for 2D data stream (timesteps, n_sensors) The format shape for 3D data stream (num_cycles, timesteps , n_sensors)

Parameters

- **quantities** (*Union[List, DataFrame, np.ndarray]*) – Measured quantities such as sensors readings.

- **target** (*Optional[Union[List, DataFrame, np.ndarray]]*) – Target label in the context of machine learning. This can be Remaining Useful Life in predictive maintenance application. Note this can be an unobservable variable in real-time and applies only for validation during offline analysis.
- **time** (*Optional[Union[List, DataFrame, np.ndarray]]*) – dtype can be either float or datetime64 to indicate the time when the quantities were measured.

set_generator_function(*generator_function: Callable = None*, *sfreq: int = None*, ***kwargs*)
Sets the data source to a generator function. By default, this function resorts to a sine wave generator function. Initialisation of the generator's parameters should be done here such as setting the sampling frequency and wave frequency. For setting it with a dataset instead, see *set_data_source()*.

Parameters

- **generator_function** (*Callable*) – A generator function which takes in at least one argument time which will be used in *next_sample()*. Parameters of the function can be fixed by providing additional arguments such as the wave frequency.
- **sfreq** (*int*) – Sampling frequency.
- ****kwargs** (*Any*) – Any additional keyword arguments to be supplied to the generator function. The **kwargs will be saved as *_generator_parameters*. The generator function call for every sample will be supplied with the **generator_parameters.

set_metadata(*device_id: str, time_name: str, time_unit: str, quantity_names: Union[str, Tuple[str, ...]], quantity_units: Union[str, Tuple[str, ...]], misc: Optional[Any] = None*)
Set the quantities metadata as a MetaData object

Details you find in the time_series_metadata.scheme.MetaData documentation.

Parameters

- **device_id** (*str*) – Name of the represented generator
- **time_name** (*str*) – Name for the time dimension
- **time_unit** (*str*) – Unit for the time
- **quantity_names** (*iterable of str or str*) – A string or an iterable of names of the represented quantities' values
- **quantity_units** (*iterable of str or str*) – An iterable of units for the quantities' values
- **misc** (*Any, optional*) – This parameter can take any additional metadata which will be handed over to the corresponding attribute of the created Metadata object

class agentMET4FOF.streams.**SineGenerator**(*sfreq=500*, *sine_freq=5*)
Built-in class of sine wave generator which inherits all methods and attributes from *DataStreamMET4FOF*. *sine_wave_function()* is a custom defined function which has a required keyword time as argument and any number of optional additional arguments (e.g F) to be supplied to the *DataStreamMET4FOF.set_generator_function()*.

Parameters

- **sfreq** (*int*) – sampling frequency which determines the time step when *next_sample()* is called
- **sine_freq** (*float*) – frequency of wave function

sine_wave_function(*time*, *sine_freq=50*)
A simple sine wave generator

agentMET4FOF.streams.**extract_x_y**(*message*)

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

1. tuple - (x,y)
2. dict - {'x':x_data,'y':y_data}

Handle data structures of dictionary to extract features & target

CHAPTER 24

agentMET4FOF metrologically enabled agents

class agentMET4FOF.metrological_agents.**MetrologicalAgent**(*name=''*, *host=None*, *serializer=None*, *transport=None*, *attributes=None*, *backend='osbrain'*, *mesa_model=None*)

_input_data = None

Input dictionary of all incoming data including metadata:

```
dict like {
    <from>: {
        "buffer": TimeSeriesBuffer(maxlen=buffer_size),
        "metadata": MetaData(**kwargs).metadata,
    }
```

_output_data = None

Output dictionary of all outgoing data including metadata:

```
dict like {
    <from>: {
        "buffer": TimeSeriesBuffer(maxlen=buffer_size),
        "metadata": MetaData(**kwargs).metadata,
    }
```

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(*input_data_maxlen=25*, *output_data_maxlen=25*)

User provided function to initialize parameters of choice.

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}.

class agentMET4FOF.metrological_agents.**MetrologicalAgentBuffer**(*buffer_size: int = 1000*)
Buffer class which is instantiated in every metrological agent to store data

This buffer is necessary to handle multiple inputs coming from agents.

We can access the buffer like a dict with exposed functions such as .values(), .keys() and .items(). The actual dict object is stored in the attribute *buffer*. The list in *supported_datatypes* contains one more element for metrological agents, namely TimeSeriesBuffer.

_concatenate(*iterable: time_series_buffer.buffer.TimeSeriesBuffer, data: Union[numpy.ndarray, list, pandas.core.frame.DataFrame], concat_axis: int = 0*) → time_series_buffer.buffer.TimeSeriesBuffer
Concatenate the given TimeSeriesBuffer with data

Add data to the TimeSeriesBuffer object.

Parameters

- **iterable** (*TimeSeriesBuffer*) – The current buffer to be concatenated with.
- **data** (*np.ndarray, DataFrame, list*) – New incoming data

Returns the original buffer with the data appended

Return type TimeSeriesBuffer

convert_single_to_tsbuffer(*single_data: Union[List[T], Tuple, numpy.ndarray]*)
Convert common data in agentMET4FOF to TimeSeriesBuffer

Parameters **single_data** (*iterable of iterables (list, tuple, np.ndarrray) with shape (N, M)*) –

- M==2 (pairs): assumed to be like (time, value)
- M==3 (triple): assumed to be like (time, value, value_unc)
- M==4 (4-tuple): assumed to be like (time, time_unc, value, value_unc)

Returns the new TimeSeriesBuffer object

Return type TimeSeriesBuffer

update(*agent_from: str, data: Union[Dict[KT, VT], List[T], Tuple, numpy.ndarray]*) → time_series_buffer.buffer.TimeSeriesBuffer

Overrides data in the buffer dict keyed by *agent_from* with value *data*

Parameters

- **agent_from** (*str*) – Name of agent sender
- **data** (*dict or iterable of iterables (list, tuple, np.ndarray) with shape (N, M)*) – the data to be stored in the metrological buffer

Returns the updated `TimeSeriesBuffer` object

Return type TimeSeriesBuffer

class agentMET4FOF.metrological_agents.**MetrologicalMonitorAgent**(*name=''*, *host=None*, *serializer=None*, *transport=None*, *attributes=None*, *backend='osbrain'*, *mesa_model=None*)

init_parameters(**args*, ***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.

CHAPTER 25

agentMET4FOF metrologically enabled streams

class agentMET4FOF.metrological_streams.**MetrologicalDataStreamMET4FOF**(*value_unc: Union[float, Iterable[float]] = 0.0*, *time_unc: Union[float, Iterable[float]] = 0.0*)

Abstract class for creating datastreams with metrological information. Inherits from the *DataStreamMET4FOF* class

To create a new *MetrologicalDataStreamMET4FOF* class, inherit this class and call *set_metadata()* in the constructor. Choose one of two types of datastreams to be created:

- from dataset file (*set_data_source()*), or
- a waveform generator function (*set_generator_function()*).

Alternatively, override the *next_sample()* function if neither option suits the application. For generator functions, *sfreq* is a required variable to be set on *init* which sets the sampling frequency and the time-step which occurs when *next_sample()* is called.

For an example implementation of using generator function, see the built-in *MetrologicalSineGenerator* class. See tutorials for more implementations.

_generator_function_unc

A generator function for the time and quantity uncertainties which takes in at least one argument `time` which will be used in *next_sample()*. The return value must be a 2-tuple of time and value uncertainties each of one of the three types:

- np.ndarray
- pandas DataFrame

- list

Type Callable

_uncertainty_parameters
Any additional keyword arguments to be supplied to the generator function. Both the calls of the value generator function and of the uncertainty generator function will be supplied with the `**_uncertainty_parameters`.

Type Dict

_default_uncertainty_generator(*time: Union[List[T], pandas.core.frame.DataFrame, numpy.ndarray], values: Union[List[T], pandas.core.frame.DataFrame, numpy.ndarray]*) → Tuple[numpy.ndarray, numpy.ndarray]
Default (standard) uncertainty generator function

Parameters

- **time** (*Union[List, DataFrame, np.ndarray]*) – timestamps
- **values** (*Union[List, DataFrame, np.ndarray]*) – values corresponding to timestamps

Returns constant time and value uncertainties each of the same shape as `time`

Return type Tuple[np.ndarray, np.ndarray]

_next_sample_generator(*batch_size: int = 1*) → numpy.ndarray
Internal method for generating a batch of samples from the generator function. Overrides *DataStreamMET4FOF._next_sample_generator()*. Adds time uncertainty `ut` and measurement uncertainty `uv` to sample

set_generator_function(*generator_function: Callable = None*, *uncertainty_generator: Callable = None*, *sfreq: int = None*, ***kwargs*) → Callable
Set value and uncertainty generators based on user-defined functions. By default, this function resorts to a sine wave generator function and a constant (zero) uncertainty. Initialisation of the generator's parameters should be done here such as setting the sampling frequency and wave frequency. For setting it with a dataset instead, see *set_data_source()*. Overwrites the default *DataStreamMET4FOF.set_generator_function()* method.

Parameters

- **generator_function** (*callable*) – A generator function which takes in at least one argument `time` which will be used in *next_sample()*.
- **uncertainty_generator** (*callable*) – An uncertainty generator function which takes in at least one argument `time` which will be used in *next_sample()*.
- **sfreq** (*int*) – Sampling frequency.
- ****kwargs** (*Optional[Dict[str, Any]]*) – Any additional keyword arguments to be supplied to the generator function. The `**kwargs` will be saved as *_uncertainty_parameters*. Both the calls of the value generator function and of the uncertainty generator function will be supplied with the `**uncertainty_parameters`.

Returns The uncertainty generator function

Return type Callable

time_unc
uncertainties associated with timestamps

Type Union[float, Iterable[float]]

value_unc
uncertainties associated with the values

Type Union[float, Iterable[float]]

class agentMET4FOF.metrological_streams.**MetrologicalSineGenerator**(*sfreq: int = 500, sine_freq: float = 50, device_id: str = 'SineGenerator', time_name: str = 'time', time_unit: str = 's', quantity_names: Union[str, Tuple[str, ...]] = 'Voltage', quantity_units: Union[str, Tuple[str, ...]] = 'V', misc: Optional[Any] = 'Simple sine wave generator', value_unc: Union[float, Iterable[float]] = 0.1, time_unc: Union[float, Iterable[float]] = 0*)

Built-in class of sine wave generator

Parameters

- **sfreq** (*int, optional*) – Sampling frequency which determines the time step when *next_sample()* is called. Defaults to 500.
- **sine_freq** (*float, optional*) – Frequency of the wave function. Defaults to 50.
- **device_id** (*str, optional*) – Name of the represented generator. Defaults to 'SineGenerator'.
- **time_name** (*str, optional*) – Name for the time dimension. Defaults to 'time'.
- **time_unit** (*str, optional*) – Unit for the time. Defaults to 's'.

- **quantity_names** (*iterable of str or str, optional*) – An iterable of names of the represented quantities' values. Defaults to ('Voltage')
- **quantity_units** (*iterable of str or str, optional*) – An iterable of units for the quantities' values. Defaults to ('V')
- **misc** (*Any, optional*) – This parameter can take any additional metadata which will be handed over to the corresponding attribute of the created `Metadata` object. Defaults to 'Simple sine wave generator'.
- **value_unc** (*iterable of floats or float, optional*) – standard uncertainty(ies) of the quantity values. Defaults to 0.5.
- **time_unc** (*iterable of floats or float, optional*) – standard uncertainty of the time stamps. Defaults to 0.

_sine_wave_function(*time*, *sine_freq*)

A simple sine wave generator

CHAPTER 26

agentMET4FOF dashboard

class agentMET4FOF.dashboard.Dashboard.**AgentDashboard**(*dashboard_modules=[]*, *dashboard_layouts=[]*, *dashboard_update_interval=3*, *max_monitors=10*, *ip_addr='127.0.0.1'*, *port=8050*, *agentNetwork='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

is_port_available(*ip_addr*, *_port*)

Check if desired ip and port are available.

run()

This is actually executed on calling start() and brings up the server

class agentMET4FOF.dashboard.Dashboard.**AgentDashboardProcess**(*dashboard_modules=[]*, *dashboard_layouts=[]*, *dashboard_update_interval=3*, *max_monitors=10*, *ip_addr='127.0.0.1'*, *port=8050*, *agentNetwork='127.0.0.1'*, *agent_ip_addr=3333*, *agent_port=None*)

Represents an agent dashboard for the osBrain backend

terminate()

This is shutting down the application server serving the web interface

class agentMET4FOF.dashboard.Dashboard.**AgentDashboardThread**(*dashboard_modules=[]*, *dashboard_layouts=[]*, *dashboard_update_interval=3*, *max_monitors=10*, *ip_addr='127.0.0.1'*, *port=8050*, *agentNetwork='127.0.0.1'*, *agent_ip_addr=3333*, *agent_port=None*)

Represents an agent dashboard for the Mesa backend

run()

This is actually executed on calling start() and brings up the server

terminate()

This is shutting down the application server serving the web interface

CHAPTER 27

Indices and tables

- genindex
- modindex
- search

CHAPTER 28

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

Python Module Index

a

Index

Symbols

A

B

C

D

E

G

H

S

T

U

V