

---

# **MLPro Documentations**

***Release 1.4.0***

**Detlef Arend, Steve Yuwono, Laxmikant Shrikant Baheti et al**

**Apr 29, 2024**



<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Key Features . . . . .	3
1.2	Architecture . . . . .	4
1.3	Development . . . . .	5
<b>2</b>	<b>Getting Started</b>	<b>7</b>
2.1	Installation from PyPI . . . . .	7
2.2	Installation from Anaconda . . . . .	7
2.3	Dependencies . . . . .	7
2.4	First Steps . . . . .	8
<b>3</b>	<b>MLPro-BF - Basic Functions</b>	<b>9</b>
3.1	Overview . . . . .	9
3.2	Layer 0 - Elementary Functions . . . . .	10
3.3	Layer 1 - Computation . . . . .	18
3.4	Layer 2 - Mathematics . . . . .	19
3.5	Layer 3 - Application Support . . . . .	21
3.6	Layer 4 - Machine Learning . . . . .	37
<b>4</b>	<b>MLPro-SL - Supervised Learning</b>	<b>43</b>
4.1	Overview . . . . .	43
4.2	Getting Started . . . . .	43
4.3	Adaptive Functions . . . . .	44
<b>5</b>	<b>MLPro-OA - Online Adaptivity</b>	<b>47</b>
5.1	Overview . . . . .	47
5.2	Getting Started . . . . .	47
5.3	Online Adaptive Stream Processing . . . . .	47
5.4	Online Adaptive Systems . . . . .	47
<b>6</b>	<b>MLPro-RL - Reinforcement Learning</b>	<b>49</b>
6.1	Overview . . . . .	49
6.2	Getting Started . . . . .	51
6.3	Environments . . . . .	52
6.4	Agents . . . . .	72
6.5	Scenarios . . . . .	82
6.6	Training and Tuning . . . . .	82
6.7	3rd Party Support . . . . .	85
<b>7</b>	<b>MLPro-GT - Game Theory</b>	<b>87</b>
7.1	Overview . . . . .	87

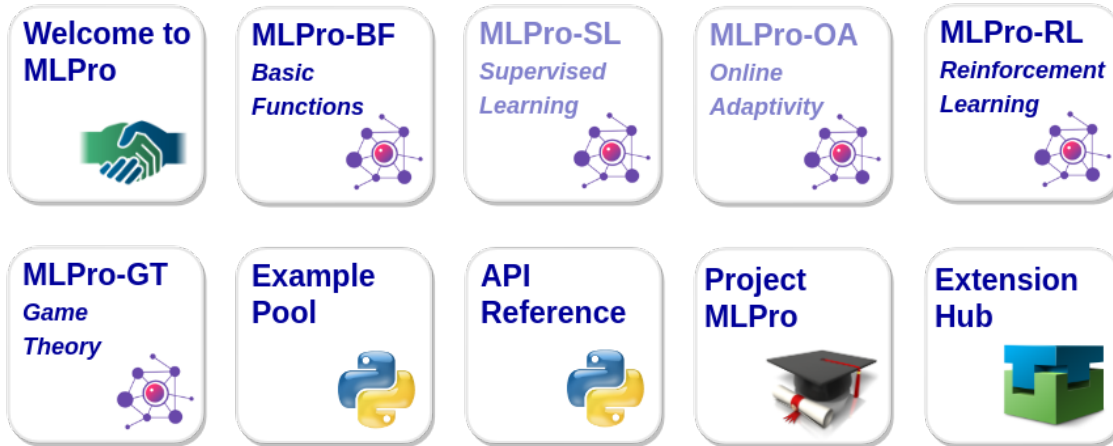
7.2	Getting Started . . . . .	88
7.3	MLPro-GT-Native - Native Games . . . . .	91
7.4	MLPro-GT-DG - Dynamic Games . . . . .	107
<b>8</b>	<b>General Information</b>	<b>119</b>
8.1	What is an MLPro Extension? . . . . .	119
8.2	How to contribute to the MLPro extension hub? . . . . .	119
8.3	Disclaimer . . . . .	120
<b>9</b>	<b>Third-Party Extensions</b>	<b>121</b>
9.1	Organizations . . . . .	121
9.2	Single Contributors . . . . .	124
<b>10</b>	<b>A1 - Example Pool</b>	<b>125</b>
10.1	MLPro-BF - Basic Functions . . . . .	125
10.2	MLPro-SL - Supervised Learning . . . . .	239
10.3	MLPro-RL - Reinforcement Learning . . . . .	239
10.4	MLPro-GT - Game Theory . . . . .	252
10.5	MLPro-OA - Online Adaptivity . . . . .	270
<b>11</b>	<b>A2 - API Reference</b>	<b>271</b>
11.1	Core Functions . . . . .	271
11.2	Pool Objects . . . . .	434
11.3	3rd Party Support . . . . .	514
<b>12</b>	<b>A3 - Project MLPro</b>	<b>519</b>
12.1	Release Notes . . . . .	519
12.2	Publications . . . . .	519
12.3	Contribution . . . . .	520
12.4	Disclaimer . . . . .	520
	<b>Python Module Index</b>	<b>521</b>
	<b>Index</b>	<b>523</b>



Welcome to MLPro - the integrative middleware framework for standardized machine learning in Python!

### MLPro

- enables hybrid ML applications in just one framework
- provides complete process landscapes and powerful ML templates on a scientific level
- integrates a growing number of *proven ML packages*
- enables comparable and reproducible results in publications
- is open source and even commercially usable ([Apache License 2.0](#))



MLPro is also present on...

[Python Package Index](#)



[Anaconda.org](#)

[GitHub](#)



## INTRODUCTION

MLPro is a comprehensive and integrative middleware framework for standardized machine learning (ML) applications in Python. The objective is to provide processes and templates for a wide range of relevant ML sub-areas without having to forego the use of already established and proven ML frameworks such as Scikit-learn, TensorFlow, PyTorch, Optuna, etc. Rather, the latter is seamlessly integrated into the process landscapes of MLPro. By using MLPro, researchers, developers, engineers, and students can focus on their essential core tasks without having to worry about the integration/interaction of different frameworks or having to re-implement existing algorithms. MLPro is architecturally designed for extensibility and recombability, which in particular enables the creation of hybrid ML applications across different learning paradigms.

### 1.1 Key Features

The most important key features of MLPro are...

#### 1. Sub-Frameworks for various ML topics

- *MLPro-RL for Reinforcement Learning*
- *MLPro-GT for Game Theory*
- *MLPro-SL for Supervised Learning*
- *MLPro-OA for Online Adaptivity*

#### 2. Powerful substructure of overarching basic functions

- *MLPro-BF for Basic Functions*

#### 3. Extensive pool of examples

- *Example Pool*

#### 4. Integration of proven 3rd party packages

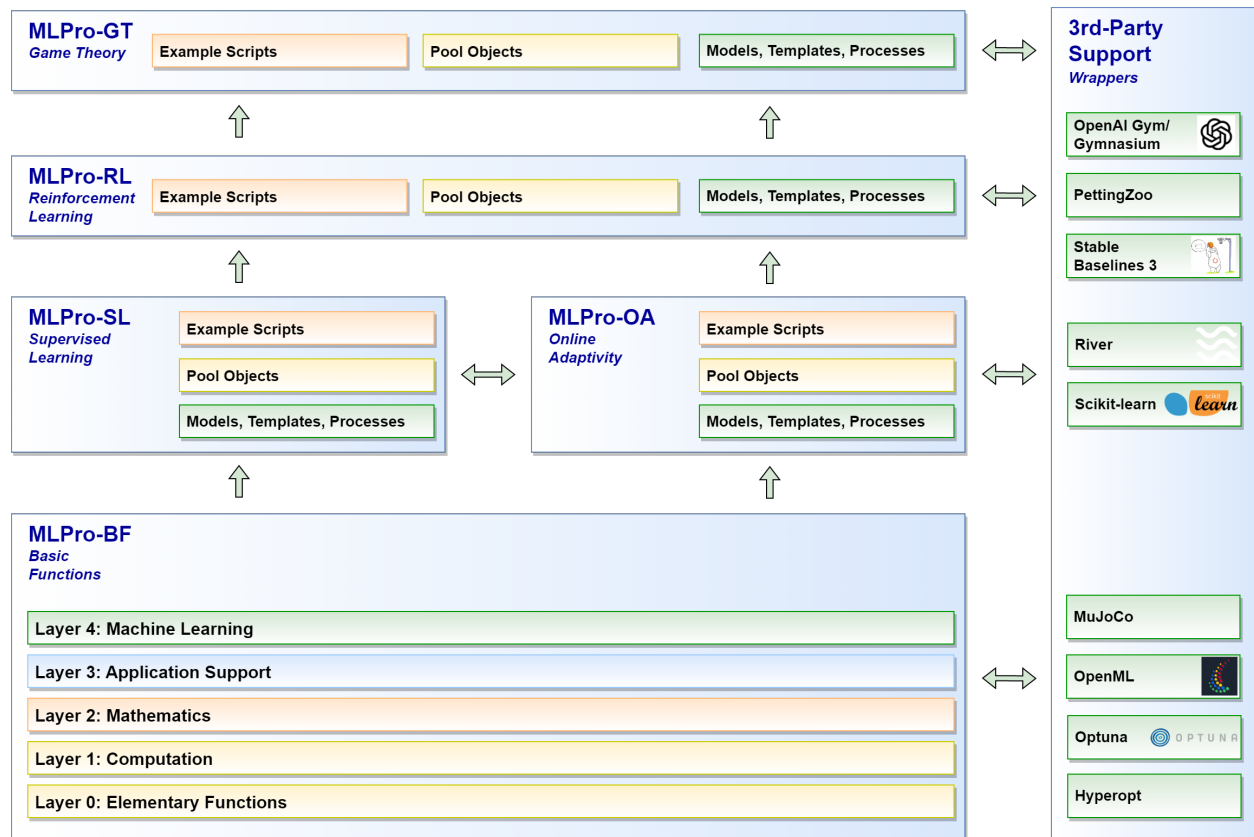
- *List of Wrappers*

#### 5. Open source, open design

## 1.2 Architecture

MLPro consists of a continuously growing number of sub-frameworks covering different areas of machine learning. These include one or more fundamental process models (e.g. the Markovian Decision Process in reinforcement learning) and appropriate service and template classes. Furthermore, each sub-framework contains a specific pool of reusable classes for algorithms, data sources, training subjects, etc. Numerous sample programs for self-study complete the scope.

The sub-frameworks mentioned are in turn based on an overarching layer of basic functions. This is a common and obvious approach. What is special about MLPro, however, is the scope and internal structure of this base layer. A spectrum of elementary functions for logging and plotting through multitasking and numerics to the basics of machine learning is covered in a hierarchy of sub-layers that build on one another. This is also the key to the far-reaching recombination of higher functions of MLPro. In fact, with each new feature, we consider how deeply we can incorporate it into MLPro. The deeper the level, the more universal the usability, and thus the range within MLPro.



### 1.2.1 Standardized Machine Learning

A special feature of MLPro is that machine learning standards are already defined in the basic functions. Templates for adaptive models and their hyperparameters as well as for executable ML scenarios are introduced in the top layer of MLPro-BF. Furthermore, standards for training and hyperparameter tuning are defined. These basic ML elements are reused and specifically extended in all higher sub-frameworks. On the one hand, this facilitates the creation of new sub-frameworks and, on the other hand, the recombination of higher functions from MLPro in your hybrid ML applications.

**Learn more:** *Basics of Machine Learning*

### 1.2.2 Example Pool

Numerous executable example programs (we call them “howtos”) illustrate the essential functions of MLPro. They are also used for validation and are therefore an integral part of our automatic unit tests. With this, we ensure two things: the operability of all howtos and thus also the operability of the demonstrated functionalities (tdd - test-driven development).

**Learn more:** *Example Pool*

### 1.2.3 Third Party Support

MLPro integrates an increasing number of selected ML packages into its process landscapes. This is done at different levels of MLPro using so-called wrapper classes that are compatible with the corresponding MLPro classes.

**Learn more:** *Wrappers*

### 1.2.4 Real-World Applications in Focus

MLPro was designed not only for simulations but for use in real-world applications. To this end, various basic functions have been implemented that make diagnostics easier and make optimal use of the available system resources. These are for example

- Detailed logging
- Precise time management of simulated and real processes on a microsecond time scale
- Creation of detailed training data files (ASCII/CSV)
- Multithreading/multiprocessing

In addition, powerful templates for state-based systems are provided. They allow the standardized implementation of your systems, which can then be controlled, for example, by adaptive controllers based on reinforcement learning or game theory. Furthermore, a wrapper for the popular physics engine [MuJoCo](#) is provided, which can be used for the simulation and visualization of externally designed system models. The MLPro templates are also prepared for connection to industrial components like controllers, sensors, and actuators.

**Learn more**

- *Elementary Functions*
- *Computation*
- *State-based Systems*

## 1.3 Development

MLPro is developed at the [South Westphalia University of Applied Sciences, Germany](#) in the [Department for Electrical Power Engineering](#) in the [Lab for Automation Technology and Learning Systems](#) and is freely available to all interested users from research and development as industry and economy.

The development team consistently applies the following principles:

- **Quality first**  
We aim to provide ML functionalities at the highest possible level. We put these up for discussion in scientific *publications*. Open feedback and suggestions for improvement are always welcome.

- **Design first**

In MLPro, new functions are not created in the code editor but in a class diagram. We provide the latter in the *API Reference*. A colour system documents the respective development status.

- **Clean Code Paradigm**

We firmly believe that a clearly structured and legible source code has a significant influence on both the acceptance and the life cycle of software. Anyone who opens any source code of MLPro knows immediately what we mean :-)

### 1.3.1 Customer Extensions

Of course, frameworks like MLPro are made to reuse their functions in own applications. That's why we put a lot of effort into design and documentation to create powerful and understandable templates and related example programs. The following notes are intended to help software developers to interpret and use them correctly.

There are essentially three types of classes in the MLPro framework:

- **Property classes**

These are classes that standardize certain properties and pass them on to child classes through inheritance. These classes are primarily found in the lower layers of MLPro and are not intended for direct use in your own applications. Nevertheless, they can of course be used in your own classes to maintain compatibility and integrity with MLPro. Examples can be found in *Basic Functions, Layer 0* among others.

- **Process classes**

These classes provide higher level application functions such as training or running a model. They are primarily found in the higher sub-frameworks for machine learning. Customer extensions are not provided here.

- **Template classes**

In order to be able to implement your own algorithms, models, data objects, systems, etc. in compliance with the MLPro standards, numerous template classes are provided on different levels. These in turn contain special **custom methods** that are intended for your own adjustments. These are explicitly identified in the *API Reference* both in the description of the classes and methods and in the associated class diagram. It should be noted here that custom methods are often inherited from parent classes (e.g. property classes). It is therefore recommended to follow the inheritance lines of template classes.

#### Learn more

- *Example Pool*
- *API Reference*

## GETTING STARTED

### 2.1 Installation from PyPI

MLPro is listed in the [Python Package Index \(PyPI\)](#) and can be installed using the package installer for Python (pip) in two variants:

- **Without any dependencies**

The following command installs the latest version of MLPro.

```
pip install mlpro
```

Additional packages may need to be installed manually (depending on the functionalities you intend to use).

- **Full installation with all dependencies**

There is also an option to automatically install MLPro and all depending packages in validated versions (see Sub-section *Dependencies* below). This option will ensure that all the functionalities of MLPro, including wrappers and examples, work appropriately out of the box.

```
pip install mlpro[full]
```

### 2.2 Installation from Anaconda

MLPro is also available on [Anaconda](#) and can be installed with the following command:

```
conda install -c mlpro mlpro
```

### 2.3 Dependencies

The table below shows all packages that MLPro has dependencies on. Additionally, the versions with which MLPro is compatible are listed. Since we cannot influence incompatible changes on dependent packages, we unfortunately cannot rule out the possibility of problems occurring with different versions. We review and update the list with each new release.

Which packages are actually required depends on the functionalities of MLPro that are used.

## 2.4 First Steps

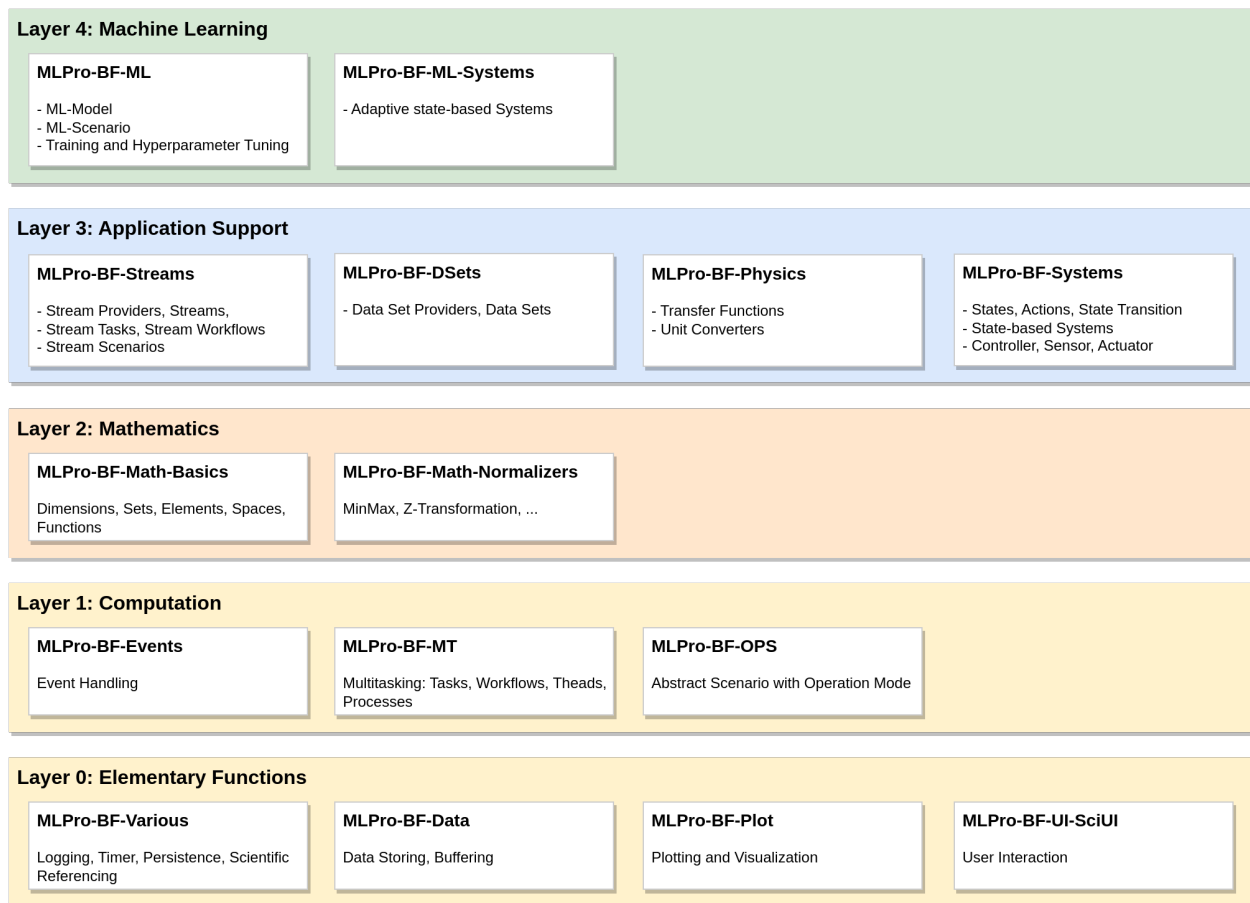
The easiest way to become familiar with the concepts and functions of MLPro is to browse through the numerous *example programs*. We can also recommend taking a closer look at the *key features* of MLPro and following the links.



## MLPRO-BF - BASIC FUNCTIONS

### 3.1 Overview

MLPro has an extensive substructure of comprehensive basic functionalities, which are combined in the sub-framework MLPro-BF. This is organized in a total of five layers that build on one another, as shown in the following figure:



The lowest *Layer 0: Elementary Functions* provides a collection of functions for logging, time measurement in simulated or real processes, persistence and data management, plotting, etc. It also contains a framework for interactive GUI applications.

The next *Layer 1: Computation* consists of various functionalities related to computation topics, such as event handling and multitasking. Furthermore, it introduces an abstract runtime scenario with an operation mode (simulation or real

operation). This is one of the key concepts in MLPro to support real applications. It is reused and specialized at higher levels.

On top of this, *Layer 2: Mathematics* introduces elementary mathematical objects like dimensions, sets and elements, metric spaces, and functions. Furthermore, numeric algorithms for data normalization etc. are included.

*Layer 3: Application Support* prepares the connection to real applications. It introduces powerful systematics for stream data processing/visualization and state-based systems that are, in turn, prepared for communication with real hardware components like sensors and actuators.

The top *Layer 4: Machine Learning* of MLPro-BF specifies fundamental standards for machine learning. All higher ML-related sub-frameworks reuse and specialize them. Topics like hyperparameters, adaptive models, and their training and tuning in ML scenarios are handled here.

## 3.2 Layer 0 - Elementary Functions

This lowest layer of MLPro-BF provides elementary functionalities for the following topics:

### 3.2.1 Logging

MLPro makes extensive use of a textual logging mechanism provided by the property class **Log**. This class centralizes and standardizes log outputs by adding appropriate methods to child classes by inheritance. It can be accessed as follows:

```
from mlpro.bf.various import Log
```

The log structure is divided into:

- Date : Current date in format year-month-day
- Time : Current time in format hour:minute:second.microsecond
- Log Type : The type of logging
- Log Text : The text that is being logged to the console

2022-09-02 14:19:26.962437 W Demo class MyClass: Something is weird...



An example of logging functionality is shown in the figure below.

```
2022-09-02 14:19:26.962434 I Demo class MyClass: Let me tell you what's going on...
2022-09-02 14:19:26.962437 W Demo class MyClass: Something is weird...
2022-09-02 14:19:26.962500 E Demo class MyClass: And here something failed...
2022-09-02 14:19:26.962505 I Demo class MyClass: But don't worry. Everything is fine. It's just a demo:)
2022-09-02 14:19:26.962508 S Demo class MyClass: This method terminated successfully!
```

In the figure above, it is shown that each log type has its own text color. Each log type is assigned to a color.

Here is the list of the color:

- Information (I) : White
- Warning (W) : Yellow
- Error (E) : Red

- Success (S) : Green

In the initialization, the logging level of object needs to be defined. By default, it will log all the informations; Information, Warning, Error, and Success. The user can specify which information that needs to be logged. The user can also change the logging level after the initialization with `switch_logging(p_logging)`.

**The following are the identifier for `p_logging`:**

- `Log.C_LOG_ALL` : Information, Warning, Error, Success
- `Log.C_LOG_WE` : Warning, Error
- `Log.C_LOG_E` : Error
- `Log.C_LOG_NOTHING` : Nothing

To log an information, function `log(p_type, *pargs)` needs to be called. The `p_type` is the type of logging. In the `*pargs`, the user can put the information in tuple.

**Here is the list for `p_type`:**

- `Log.C_LOG_TYPE_I` : Information
- `Log.C_LOG_TYPE_W` : Warning
- `Log.C_LOG_TYPE_E` : Error
- `Log.C_LOG_TYPE_S` : Success

```
from mlpro.bf.various import Log

class MyRandomClass(Log):

    def __init__(self, p_logging=True):
        # The constructor of class Log initializes the internal logging
        super().__init__(p_logging=p_logging)

    def random_method(self):
        self.log(self.C_LOG_TYPE_I, "Hi, I am information!")
        self.log(self.C_LOG_TYPE_W, "Hi, I am warning!")
        self.log(self.C_LOG_TYPE_E, "Hi, I am error!")
        self.log(self.C_LOG_TYPE_S, "Hi, I am success!")

if __name__ == "__main__":
    # Initilaization
    my_random_class = MyRandomClass(p_logging=Log.C_LOG_ALL)

    # Switch the logging level to Log.C_LOG_WE
    my_random_class.switch_logging(Log.C_LOG_WE)

    # Switch the logging level to Log.C_LOG_E
    my_random_class.switch_logging(Log.C_LOG_E)

    # Switch the logging level to Log.C_LOG_NOTHING
    my_random_class.switch_logging(Log.C_LOG_NOTHING)
```

## Cross Reference

- *Howto BF-001: Logging*

- [API Reference](#)

### 3.2.2 Time Measurement

MLPro provides an internal timing mechanism that is introduced by class property **Timer**. This class uses the built-in python package, namely **datetime**, to deal with the time management system. This class also has a simple lap management, in which each time the maximum number of laps `C_LAP_LIMIT` is reached, then the lap counter restarts to 0. Timer class can be accessed as follows:

```
from mlpro.bf.various import Timer
```

The time measurement can cover two different modes, such as:

- `C_MODE_REAL` : real-time mode
- `C_MODE_VIRTUAL` : virtual time mode

The following are the functionalities of the timer:

- `reset` : to reset timer
- `get_time` : to get the actual time
- `get_lap_time` : to get the actual lap time
- `get_lap_id` : to get an id of an actual lap
- `add_time` : to add actual time, which is specifically for virtual mode
- `finish_lap` : to end the current lap

Cross Reference

- [Howto BF-002: Timer](#)
- [API Reference](#)

### 3.2.3 Persistence

MLPro already introduces the property class **Persistent** on the lowest level of its basic functions. This class provides the two public methods `load()` and `save()` for loading and saving objects as files. As usual in Python, the standard module **pickle** is internally used for de-/serialization. More specifically, the **dill** extension is used, which extends **pickle**'s scope of natively serializable data types. Actually, neither **pickle** nor **dill** can natively serialize all types of data. For example, third-party packages such as Pygame, PyTorch, or MuJoCo provide data structures that cannot be natively serialized with **pickle/dill**. In order to be able to use such non-serializable data objects in a consistent persistence strategy, the Python standard offers the two custom methods

```
__setstate__(self, state:dict)
```

and

```
__getstate__(self) -> dict
```

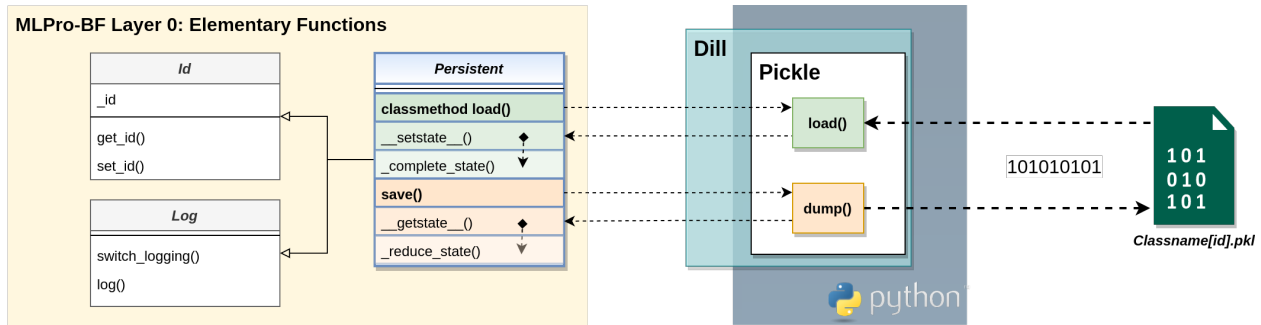
Classes that implement these methods can exclude non-serializable content from de-/serialization process and handle it appropriately. The **Persistent** class implements both of these methods and defines two new custom methods

```
_complete_state(self, p_path:str, p_os_sep:str, p_filename_stub:str)
```

and

```
_reduce_state(self, p_state:dict, p_path:str, p_os_sep:str, p_filename_stub:str)
```

These, in turn, provide the correct file path and a standard file name (without an extension, hence “stub”) to ensure the integrity of all files related to the object. The file name is generated and consists of the name of the class and the unique ID of the object to be de-/serialized. The figure below shows the correlations between the Python standard module **pickle**, it's extension **dill** and MLPro's class **Persistent**:



### Saving Objects

Saving an object works basically like this:

```
from mlpro.bf.various import Persistent

class MyClass (Persistent):

    def _complete_state(self, p_path:str, p_os_sep:str, p_filename_stub:str):
        """
        To be called by Python standard method __setstate__() during deserialization...
        """
        pass

    def _reduce_state(self, p_state:dict, p_path:str, p_os_sep:str, p_filename_stub:str):
        """
        To be called by Python standard method __getstate__() during serialization...
        """
        pass

mc = MyClass()
mc.save( p_path='c:\tmp')
```

If the file name itself is omitted from the **save()** method, MLPro generates a default file name as previously described. For example:

```
MyClass[8aa41da2-0748-4cd0-9025-55e9d9d9a131].pkl
```

In the custom method **\_reduce\_state()**, non-serializable parts of **MyClass** can now be saved in other files with the same name and directory but specific file extension, such as

```
MyClass[8aa41da2-0748-4cd0-9025-55e9d9d9a131].csv
```

### Loading Objects

The **load()** method of the **Persistent** class is defined as a class method because a related object is only generated by loading. So it's essential to know which class is used to load a previously saved file:

```
mc = MyClass.load( p_path='c:\tmp', p_filename='MyClass[8aa41da2-0748-4cd0-9025-  
↪55e9d9d9a131].csv')
```

In this case, the custom method `_complete_state()` of class **MyClass** is used to automatically load additional content into the object.

### Version of Persistence

The **Persistent** class also contains a class attribute

```
C_PERSISTENCE_VERSION : str = '1.0.0'
```

that labels the current implementation of persistence with a unique version. The **load()** method compares the version of the loading class and the object to be loaded and denies file access if the versions differ.

---

**Note:** The version of a child class of **Persistent** should be increased in case of incompatible changes on the methods `_complete_state()` or `_reduce_state()`.

---

### Cross Reference

- [Howto BF-005: Persistence](#)
- [API Reference BF-VARIOUS - Various Functions](#)
- [Python Documentation: Python object serialization](#)
- [Python Documentation: Persistence of external objects](#)
- [Dill: Python extension for pickle](#)

## 3.2.4 Data Management

Data management in a framework is extremely important, which mostly refers to the organization, storage, and retrieval of data within the framework. In MLPro, our team also provides such functionalities as saving data, loading data, storing data, creating a buffer, and plotting data. This involves defining a data model that describes the structure and relationships between data elements, implementing mechanisms for storing and retrieving data, and managing data consistency and integrity. A well-designed data management system is essential for the efficient and effective processing of data within the framework.

On this page, the data management within the MLPro framework is explained. However, an explanation of the plotting functionality is provided in the *next subchapter* of this documentation. The related data management classes can be accessed as follows:

```
from mlpro.bf.data import *
```

In general, there are two main functionalities of data management in MLPro:

#### 1) Data Storing

The second possibility is to store a bunch of data in MLPro's **DataStoring** class with three different layers, as follows:

- **Layer 1 - Data Names** : the labels or the feature names of the data.
- **Layer 2 - Frames** : the frames can be added to each label or feature name. If none, then the frame id can be set to '0' all the time.
- **Layer 3 - Values** : the values can be added to each frame with a specific label or feature name.

Therefore, to add value to the data storage, the users can use `DataStoring.memorize(p_variable, p_frame_id, p_value)`, in which `p_variable`, `p_frame_id`, `p_value` refer to the feature name, the frame id, and the added value respectively.

For better understanding : [Howto BF-003: Store and plot data](#)

## 2) Buffering

The other data handling functionality is buffering by MLPro's **Buffer** class. The buffer is an important component in machine learning and online learning areas, where a number of data have to be stored, updated frequently, and also used for data sampling purposes. The following shows what the users can do with the buffer and how to define the buffer:

- Redefining `Buffer._gen_sample_ind(self, p_num: int)` to set up a method of sampling data from the buffer or optionally simply using **BufferRnd** class with random sampling functionality.
- Instantiating a buffer with a defined maximum buffer size.
- Storing the target values with specific feature names using **BufferElement**.
- Adding the buffer element to the buffer.
- Optional functionalities:
  - Checking whether the buffer is full.
  - Obtaining all data from the buffer.
  - Sampling from the buffer.
  - Clearing the buffer.

For better understanding : [Howto BF-004: Buffers](#)

## Cross Reference

- [Howto BF-003: Store and plot data](#)
- [Howto BF-004: Buffers](#)
- [API Reference](#)

## 3.2.5 Plotting and Visualization

In order to be able to visualize processes on different levels in a standardized way, MLPro provides a property class **Plottable**. This inherits to higher classes (custom) methods for initializing and updating plot output during execution. There are three different views:

- 2-dimensional plot output
- 3-dimensional plot output
- n-dimensional plot output

Additional parameters can be set using the **PlotSettings** class.

MLPro and in particular the **Plottable** class is intended for plot outputs with the standard package **Matplotlib** in connection with the output backend **TkAgg**. In this combination, a good user experience is made possible. In principle, however, other packages can also be used for visualization.

## Cross Reference

- [API Reference BF-PLOT - Plotting and Visualization](#)
- [Stream Plotting](#)

### 3.2.6 Scientific Referencing

MLPro integrates scientific referencing in any class using a class **ScientificObject**. This class provides elementary functionality for storing scientific references. For example, when the users create a custom reinforcement learning policy or a custom environment, then the users can simply inherit ScientificObject class and add a scientific reference to the related elements. This class can be accessed as follows:

```
from mlpro.bf.various import ScientificObject
```

MLPro provides various forms of scientific references, which are:

- C\_SCIREF\_TYPE\_NONE : None
- C\_SCIREF\_TYPE\_ARTICLE : Journal Article
- C\_SCIREF\_TYPE\_BOOK : Book
- C\_SCIREF\_TYPE\_ONLINE : Online
- C\_SCIREF\_TYPE\_PROCEEDINGS : Proceedings
- C\_SCIREF\_TYPE\_TECHREPORT : Technical Report
- C\_SCIREF\_TYPE\_UNPUBLISHED : Unpublished

After selecting the type of reference, the users can add more details, such as authors, titles, volume, DOI, and many more.

The type and detail of the related scientific reference in a class can be initialized, as follows:

```
from mlpro.bf.various import ScientificObject

class MyClass(ScientificObject):

    def __init__(self):
        self.C_SCIREF_TYPE = self.C_SCIREF_TYPE_ARTICLE
        self.C_SCIREF_AUTHOR = "Max Mustermann"
        self.C_SCIREF_TITLE = "Analysis of MLPro"
        self.C_SCIREF_JOURNAL = "My Journal"
        self.C_SCIREF_YEAR = "2023"
        self.C_SCIREF_MONTH = "01"
        self.C_SCIREF_DAY = "01"
        self.C_SCIREF_VOLUME = "01"
        self.C_SCIREF_DOI = "10.XXXX"
```

MLPro team has added a citing functionality. Therefore, the users can obtain the citation of the specific class in the form of BibTeX, as follows:

```
@article{CitekeyArticle,
author = {Max Mustermann},
title = {Analysis of MLPro},
journal = {My Journal},
volume = {01},
year = {2023},
month = {01},
day = {01},
doi = {10.XXXX}
}
```



## Cross Reference

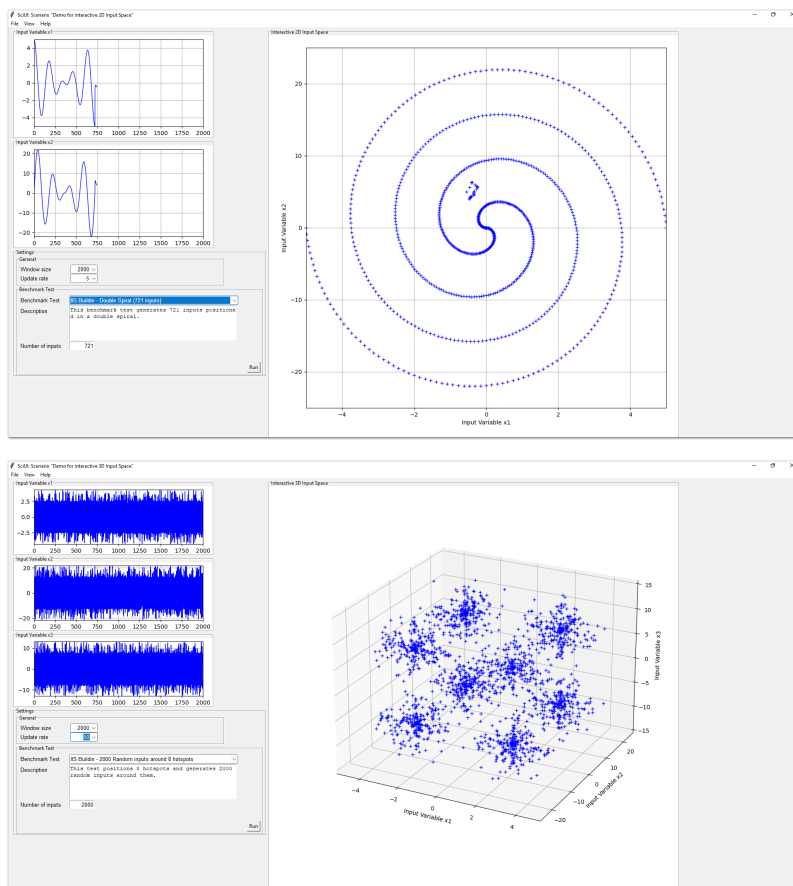
- [API Reference](#)

## 3.2.7 User Interaction

MLPro-BF provides a framework called SciUI (Scientific User Interface) for creating and running interactive ML applications for graphical validation, presentation and education purposes.

Key features are:

- Platform-independend framework for creation of own UI scenarios
- Ready-to-run application „SciUI“ auto-detects and starts own scenarios
- Focus on real-time visualization and interaction
- Based on Python standards Tkinter and Matplotlib



## Cross Reference

- [Howto BF-UI-001: SciUI - Reuse of interactive 2D/3D Input Space](#)
- [Howto BF-UI-002: SciUI - Reinforcement Learning Cockpit](#)

## 3.3 Layer 1 - Computation

This layer introduces various techniques for the efficient execution of higher functionalities of MLPro. In particular, it is dedicated to these topics:

### 3.3.1 Event Handling

Event handling is a widely used standard technique in software development. And that's how it found its way into MLPro. The mechanism is inherited by higher classes in the form of the property class **EventHandler**. This class allows event handler methods to be registered and events to be triggered, which in turn call registered handlers. An object of type **Event** is passed to each handler. This contains further information about the context of the event.

The event handling functionality is used extensively in MLPro. A large number of higher classes use this mechanism. Based on this, even an event-oriented adaptation mechanism is cultivated in *Layer 4 - Machine Learning*.

#### Cross Reference

- *Howto BF-EH-001: Event Handling*
- *API Reference BF-EVENTS - Event Handling*

### 3.3.2 Multitasking

In MLPro, the two essential techniques **Multithreading** and **Multiprocessing** for the asynchronous execution of program parts are summarized under multitasking. Both techniques allow the use of several or even all cores of the CPU(s), which can significantly increase the execution speed of the respective program. The basic prerequisite for this is, of course, that significant parts of the program can actually be executed in parallel.

While with multithreading program parts are executed in parallel within the same process, with multiprocessing they are started in separate processes. The latter uses computing resources more effectively but requires more administrative effort on the part of the operating system. Therefore, it is hard to say in advance which of the two techniques is the better choice for a specific implementation. What can be said, however, is that multithreading is the more straightforward technique to use with good acceleration of parallel programs. Multiprocessing requires a little more detailed knowledge of the internal mechanisms of the runtime environment. It unfolds its strengths, in particular with long-running parallel program parts, since the administrative overhead is not so significant here. The pros and cons of multithreading and multiprocessing can be excellently discussed. For this purpose, reference is made to relevant sources on the Internet.

The details of the multitasking implemented in MLPro are explained in more detail below:

#### Range

In the context of multitasking, the range describes the degree of parallelism of a program function. There are three different degrees:

- Synchronous/serial
- Asynchronous/parallel using multithreading
- Asynchronous/parallel using multiprocessing

#### Asynchronous execution of methods

In order to enable the asynchronous execution of methods of a class, MLPro provides the property class **Async**. In particular, this includes the method `_start_async()`, which allows the execution of another method in a separate thread or process.

#### Tasks and Workflows

A fundamental and consistently used concept in MLPro is that of tasks and workflows. A task is a class that can be executed in one of the three possible ranges mentioned above. Tasks can in turn be grouped into workflows. Any **directed graphs** of tasks can be set up and processed massively parallel via corresponding predecessor relationships. Successor tasks are informed about the termination of their predecessors via event technology. The **Task** and **Workflow** classes of the same name have once again been implemented as property classes. They are consistently reused in higher functions through inheritance. Some examples are *Stream Processing* and the *Adaptive Workflows*.

### Gap under MacOS

At the time of the creation of MLPro, a technical problem related to multiprocessing occurred on Apple computers under MacOS. This is documented at [Race condition when using multiprocessing BaseManager and Pool in Python3](#). It is recommended to check MLPro functionalities in multiprocessing mode on MacOS-based computers very carefully and, if in doubt, to use multithreading.

### Cross Reference

- [Howto BF-MT-001: Multitasking - Parallel Algorithms](#)
- [Howto BF-MT-002: Multitasking - Tasks and Workflows](#)
- [API Reference BF-MT - Multitasking](#)

## 3.3.3 Operations

In this module, classes are made available that are required for the operative execution of higher functions of MLPro. In particular, the **ScenarioBase** class is introduced here, which serves as an abstract template for various scenarios, such as *Stream Scenarios* or *ML Scenarios*. In this respect, the scenario in MLPro is one of the fundamental concepts, which already introduces the following properties at this low level:

- Runtime mode (simulation or real operation)
- Execution of cycles
- Internal Timer-Management
- Persistence

All scenario classes in MLPro are ultimately template classes for implementing your own concrete applications. Therefore, special attention should be paid to the custom methods that are already introduced here.

### Cross Reference

- [API Reference BF-OPS - Operations](#)
- [BF-Various - Persistence](#)

## 3.4 Layer 2 - Mathematics

Mathematics is an integral part of many areas, particularly in the fields of data analysis, machine learning, and system simulation. MLPro-BF-MATH provides the basic functions of mathematics, which become the foundation for various computational tasks. The basic functions of mathematics simplify the users of defining, conserving, and processing numeric data at a low level.

First, four main components are introduced in MLPro-BF-MATH to define a bunch of numeric data, such as:

### a) Dimension

An object specifies properties of a dimension, including the name, the unit, the data type, the boundaries, and many more. Dimension in MLPro is not limited to real, natural, and integer numbers, but can also handle a big data object (e.g. images, point cloud, etc.).

b) **Set**

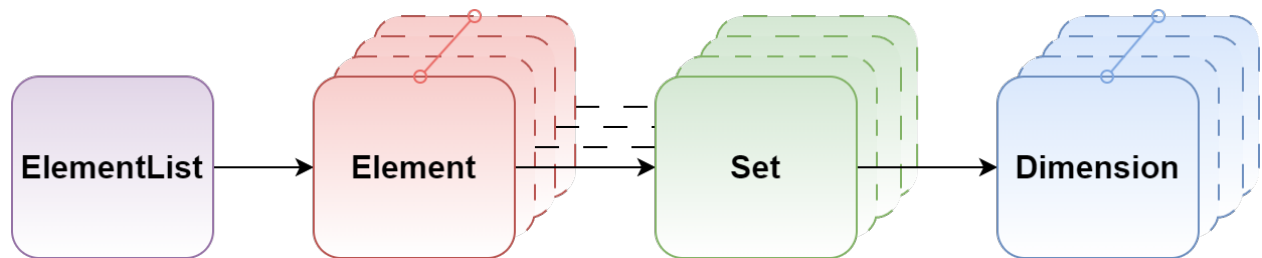
Set is a collection of dimensions that can also be described as a multivariate set in a mathematical sense. There are interesting functionalities, for instance, adding a new dimension to the set, getting information on the actual dimension in the set, and spawning a new object with a subset of dimensions.

c) **Element**

Element of a multivariate set. Each element represents a set, where the values of each component (Dimension object) can now be set.

d) **ElementList**

ElementList consists of a list of Element objects.



Second, MLPro also provides two objects to measure the distance between two elements, such as **MSpace** and **ESpace**. **MSpace** represents a metric space. The distance calculation is based on the metric of the space. **ESpace** represents a Euclidean space. The distance calculation is based on the Euclidean norm.

Third, MLPro has a **Function** class with mapping functionality. This class is intended for an elementary bi-multivariate mathematical function that maps elements of a multivariate input space to elements of a multivariate output space. With this class, it is possible to map a multivariate abscissa/input element to a multivariate ordinate/output element.

Lastly, there are more advanced functions of mathematics, as follows:

### 3.4.1 Normalization

Normalization is a process of scaling different parameters to a common scale. The way the parameters are scaled depends on the type of normalization being performed. For e.g. parameters are scaled within a range of -1 to 1 in case of a minmax normalization. MLPro's normalizer classes can be used to normalize data based on MinMax Normalization and Z-transformation. These normalizer classes can be imported by incorporating following lines in your script.

```

from mlpro.bf.math.normalizers import NormalizerMinMax
from mlpro.bf.math.normalizers import NormalizerZTransform
  
```

Both normalizers store the parameters required for normalization based on the data provided for normalization. MLPro also provides the possibility to set/update the parameters when required, based on data instances or direct parameters for e.g boundaries for MinMax normalizers.

**Both the normalizers provide following operations:**

- **Normalize** : Normalize a given data element based on the set parameters.
- **Denormalize** : Denormalize a given data element based on the set parameters.
- **Update Parameters** : Upadte the normalization parameters based on data characteristics such as boundaries or statistical properties.
- **Renormalize** : MLPro's normalizers also provide the possibility to renormalize the previously normalized data elements on new normalization parameters.

Cross Reference

- [Howto BF-MATH-010: Normalizers](#)
- [API Reference](#)

3.4.2 Geometry

...

Cross Reference

- [API Reference](#)

Cross Reference

- [Howto BF-MATH-001: Dimensions, Spaces and Elements](#)
- [API Reference](#)

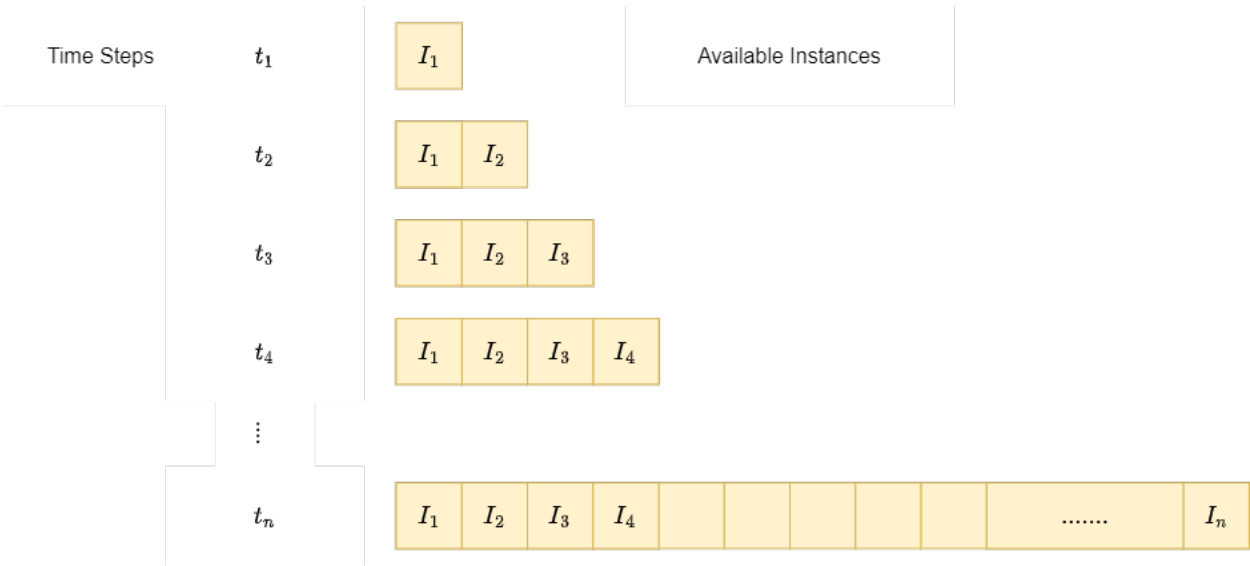
3.5 Layer 3 - Application Support

This layer deals with the provision of basic technologies to support real-world applications.

3.5.1 Stream Processing

Streams

A data stream is a live data source that delivers instances sequentially. Unlike offline datasets, data instances cannot be scanned on demand in case of streams. Data instances are only available at the order they arrive. For example, think of a live RADIO signal that delivers new data with time, where complete access to entire data is not possible.



As shown in the figure above, at every timestep, new information is available. However, the number of instances delivered at each instance and availability of historical instances depends on the type of stream and the processing task respectively.

In industrial scenarios, with more and more complex systems, the amount of live data delivered by the systems increases rapidly. This high amount of live data can be leveraged to take optimal decisions for processes. This real-time data is a data stream because of its live nature and processing such real-time data is a relevant field of data-mining and machine learning known as Stream Processing and Online Machine Learning respectively.

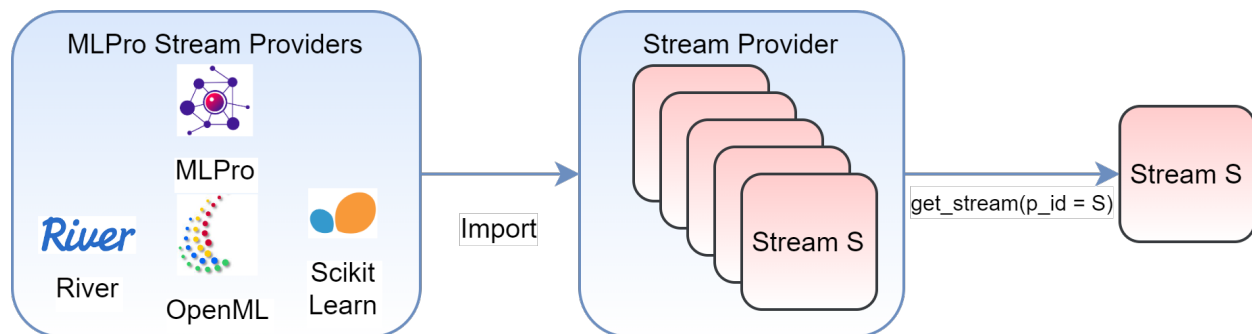
After loading the stream provider (MLPro's native stream provider for example), the list of available streams can be loaded as following:

```
# Import the stream provider class
from mlpro.bf.streams.native import NativeStreamProvider
# Create an object of the stream provider
mlpro = NativeStreamProvider()
# Get a list of streams
mlpro.get_stream_list()
```

**Learn more**

## Streams Handling

MLPro's stream module provides a stream handling and bundle of stream processing functionalities. The stream handling architecture in MLPro is as shown in the following figure:



The figure shows a collection of stream provider apis, which in turn, contain a list of corresponding stream objects. Currently, MLPro supports stream provider api for MLPro's native streams and three external data providers:

- OpenML
- River
- Scikit-Learn

## Stream Provider

Access to real-time data stream is not always possible for the purpose of testing and evaluations. MLPro's streams module provides Stream Provider functionality. A stream provider in MLPro is a data resource that provides stream objects for various operations.

MLPro's streams module provides native stream providers, that generate stream objects with user-defined parameters such as number of features and labels and pre-defined statistical properties such as feature boundaries. Currently MLPro's native stream provider supports random streams with random feature and label values. Along with native stream provider MLPro also supports data resources from popular external data resources including OpenML, ScikitLearn and River. MLPro's stream provider object accesses datasets from these resources and provide them as stream objects that imitate the sequential behaviour.

A stream provider in MLPro can be imported by including:

```
# import mlpro native stream provider
from mlpro.bf.streams.native import NativeStreamProvider
# import openml stream provider
from mlpro.wrappers.openml import WrOpenMLStreamProvider
# import river stream provider
from mlpro.wrappers.river import WrRiverStreamProvider
# import scikit learn stream provider
from mlpro.wrappers.sklearn import WrSKLearnStreamProvider
```

After loading the stream provider (MLPro's native stream provider for example), the list of available streams can be loaded as following:

```
# Import the stream provider class
from mlpro.bf.streams.native import NativeStreamProvider
# Create an object of the stream provider
mlpro = NativeStreamProvider()
# Get a list of streams
mlpro.get_stream_list()
```

## Stream

In MLPro, a stream is a special iterator object that delivers new data instances with each iteration. A stream cannot be read directly for all the instances, instead an instance is only available when requested by a workflow. An instance in MLPro consists of feature and label data for that specific instance.

From a stream provider a specific stream of interest can be accessed with a stream id:

```
mystreamobject = mlpro.get_stream(p_id = '1')
```

After accessing the stream from the stream provider, a new instance can be accessed from the data stream by iterating over it.

## Stream Instance

An instance in MLPro is a data element available at each time step, when processing a stream. An instance consists of a unique id, feature data and label data.

```
# Accessing an instance from stream
instance = next(iter(mystreamobject))

# Accessing the stream ID
id = instance.get_id()

# Accessing feature data
feature_element = instance.get_feature_data()
feature_data = feature_element.get_values()

# Accessing label data
label_element = instance.get_label_data()
label_data = label_element.get_values()
```

**Note:**

- The ids of the stream instances are managed internally by a Stream Workflow, and are also used for stream plotting functionalities. Changing instance ids might affect the performance of stream functionalities of MLPro.

**Stream Sampler**

In MLPro, a stream has an optional component, which is a stream sampler. A sampler is a component that selects a subset of instances from a continuous stream of data. The purpose of a sampler is to reduce the volume of data that needs to be processed, while still providing a representative sample of the data.

Each streaming instance is going through the **omit\_instance** method that is provided by a sampler. If the output is True, then the instance is omitted and not part of the subset of instances being sampled. Otherwise, the instance is added to the subset of instances.

A stream sampler can be attached to a stream during its instantiation or after instantiation through the public method **setup\_sampler**.

**Note:** There are several different ready-to-use samplers in the pool of objects that can be used in MLPro stream processing, including random samplers, [min-wise samplers](#), [reservoir samplers with Algorithm R](#), and more. Each type of sampler has its characteristics and is suitable for different types of data and processing scenarios.

**Cross References**

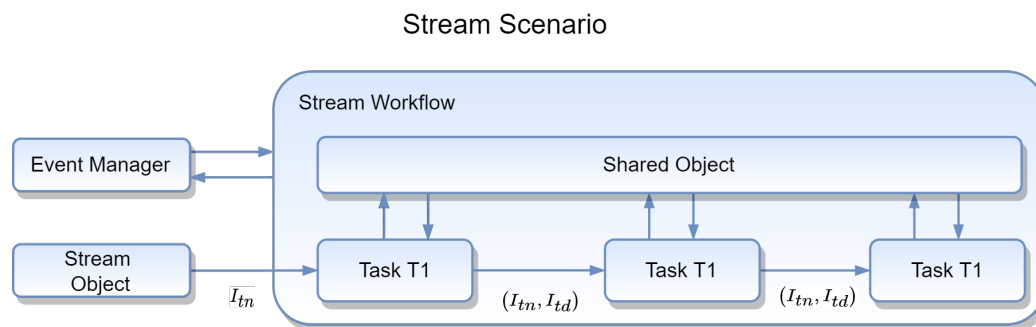
- [Howto BF-STREAMS-005: Streams Sampler](#)
- [API Reference: Streams](#)

**Cross Reference**

- [Howto BF-STREAMS-101: Basics of Streams](#)
- [API Reference: Streams](#)

**Stream Processing**

Handling streaming data sources and mining knowledge from them requires special types of processing tasks because of their live behaviour. Stream operations process new instances as they are available at every step. Along with a number of external and internal stream resources, MLPro's stream module provides processing functionalities like sliding window, rearranger, etc. specialized for streaming data.





In MLPro, streaming data is processed with a task and workflow architecture. A StreamTask is single operation performed on new stream instances and a StreamWorkflow is a list of tasks arranged sequentially with defined dependencies. **StreamTask** and **StreamWorkflow** are specialized classes inherited from MLPro's multiprocessing module. As shown in the above figure the scenario fetches new  $I_{tn}$  instances from the stream object and each task then processes a list of new instances  $I_{tn}$  and deleted/obsolete instances  $I_{td}$  as shown in the figure. The processed instances are stored in the shared object for further accessibility.

**Learn more**

## Stream Task

A StreamTask is a special stream processing task that takes a new instance as an input and delivers the processed instances as an output. A StreamTask also processes the obsolete/deleted instances from the workflow for following tasks.

StreamTask class in MLPro also provide provide plotting functionalities in 2D, 3D and nD, that plot the streaming instances by default. (A link to know more). Inherit from this class and implement the `_run(p_inst_new, p_inst_del)` method to implement custom stream tasks with inbuilt default plotting functionalities. This can be imported and used by including following:

```
#import stream models
from mlpro.bf.streams.models import StreamTask

#create a stream object
myStreamTask = Task(p_name = 'Task 1',
                    p_visualize = True,
                    p_logging = Log.C_LOG_ALL)
```

Currently MLPro provides following stream task implementations:

1. *Window*
2. *Rearranger*
3. *Deriver*

More StreamTask implementations will be available with future updates.

## Stream Workflow

A StreamWorkflow in MLPro is a list of StreamTasks arranged hierarchically with user-defined dependencies on prior tasks in the workflow. A stream workflow receives new instance of the stream from the surrounding StreamScenario object at every step.

---

**Note:** A stream workflow carries a list of new instances and deleted/obsolete instances at every run. Both new and deleted instances are forwarded to subsequent tasks to be processed.

---

**A stream workflow takes care of following functionalities:**

1. Executing the tasks inside the workflow
2. Storing task specific results in the StreamShared Object
3. Fetching and delivering new and deleted instances among different tasks as per the defined dependency

**StreamWorkflow can be imported and used as following:**

```
#import stream models
from mlpro.bf.streams.models import *

#create a stream workflow object
myStreamWorkflow = StreamWorkflow( p_name='My Workflow',
                                   p_range_max=StreamWorkflow.C_RANGE_NONE,
                                   p_visualize=True,
                                   p_logging=Log.C_LOG_ALL))
```

A stream workflow consists a list of tasks within in a defined order and instance dependency. The instances processed by a task are forwarded to it's following task. The code block below shows how to add a task to an existing stream workflow:

```
# add task myStreamTask to the workflow myStreamWorkflow
myStreamWorkflow.add_task(p_task = 'Task 1')

#create another task
myStreamTask2 = StreamTask(p_name = 'Task 1',
                           p_visualize = True,
                           p_logging = Log.C_LOG_ALL)

# add the task to the workflow with task 1 as its predecessor
myStreamWorkflow.add_task(p_task = 'Task 2', p_predecessor = 'Task 1')
```

Each workflow has a shared object that stores instances and results of the stream task that can be accessed from other tasks in the workflow. StreamWorkflow also provides default plotting functionalities in 2D, 3D and nD, that plot all the instances in the workflow. Know more about MLPro's plotting functionalities.

## Stream Scenario

A stream scenario in MLPro inherits from MLPro's scenario base class. The idea of a scenario in MLPro is to have all the elements together, required for a specific application, whether it is a training application or just a sample run. A scenario set's up the process parameters and runs the process for a given number of cycles as defined in the specific scenario implementation.

**A stream scenario consists of two main elements:**

- A stream object
- A streamtask workflow

---

**Note:** To plug these elements into the StreamScenario class, please implement the `_setup(p_mode, p_visualize, p_logging)` method of the same

---

**A StreamScenario class takes care of the following tasks in a Stream processing application:**

1. Fetching new instance at every step
2. Running the plugged in StreamWorkflow
3. Managing and updating the visualization windows
4. Storing the results of the workflow

## Cross Reference

- [Stream](#)
- [How To to be included](#)
- [API References](#)

## Stream Plotting

MLPro's streams module also provide plotting functionalities by default. The stream workflow and stream tasks can plot instances within the workflow and the task respectively. The default plotting functionality is available in 2 dimensional, 3 dimensional and N dimensional views. The plot view and specific plot properties can be set using a PlotSetting object. Below images show an example of the default plotting functionality in ND, 2D, 3D, respectively, in MLPro's streams module.

## Cross References

- [Howto BF-STREAMS-102: Tasks Workflows And Stream Scenarios](#)
- [BF-MT - Multitasking](#)
- [API Reference: Streams](#)

## Pool Objects

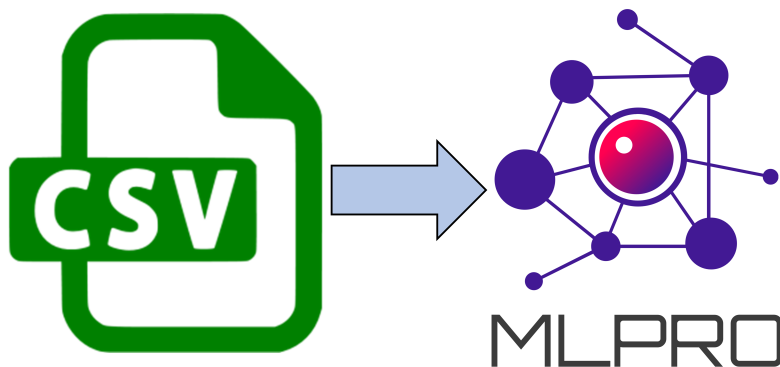
### Native Streams

Along with third party stream support, MLPro also provides a pool of native stream objects. The native streams in MLPro include the following implementations.

### Data from CSV Files

Ver. 1.1.2 (2023-04-17)

This module provides the native stream class StreamMLProCSV. This stream provides a functionality to convert csv file to a MLPro compatible stream data.



## Cross References

- [Howto BF-STREAMS-002: Accessing Data From CSV Files](#)

## Random 10-Dimensional

Ver. 1.0.0 (2022-12-13)

This module provides the native stream class `StreamMLProRnd10D`. This stream provides 1000 instances with 10-dimensional random feature data and 2-dimensional random label data.

### Cross References

- *Howto BF-STREAMS-003: Visualizing 10-dimensional Random Stream Provided By MLPro*

## 2D Double Spiral

### Cross References

- *Howto BF-STREAMS-004: Visualizing 2D Double Spiral Stream Provided By MLPro*

## Point Outliers

Ver. 1.1.0 (2024-04-26)

This module provides a multivariate benchmark stream with configurable baselines per feature and additional random point outliers.

### Cross Reference

- *Howto BF-STREAMS-005: Visualizing Multivariate Point Outlier Stream Provided By MLPro*

## Random Point Clouds (2D, 3D, ND), Static or Dynamic

Ver. 1.2.2 (2024-02-09)

This module provides the native stream classes `StreamMLProClouds`, `StreamMLProClouds2D4C1000Static`, `StreamMLProClouds3D8C2000Static`, `StreamMLProClouds2D4C5000Dynamic` and `StreamMLProClouds3D8C10000Dynamic`. These stream provides instances with `self.C_NUM_DIMENSIONS` dimensional random feature data, placed around centers (can be defined by user) which may or maynot move over time.

2D random clouds...

3D random clouds...

### Cross Reference

- *Howto BF-STREAMS-006: Visualizing Static 2D Random Point Clouds Provided By MLPro*
- *Howto BF-STREAMS-007: Visualizing Dynamic 2D Random Point Clouds Provided By MLPro*
- *Howto BF-STREAMS-008: Visualizing Static 3D Random Point Clouds Provided By MLPro*
- *Howto BF-STREAMS-009: Visualizing Dynamic 3D Random Point Clouds Provided By MLPro*
- *Howto BF-STREAMS-010: Visualizing Multivariate Random Cloud Generator in 3D Mode Provided By MLPro*

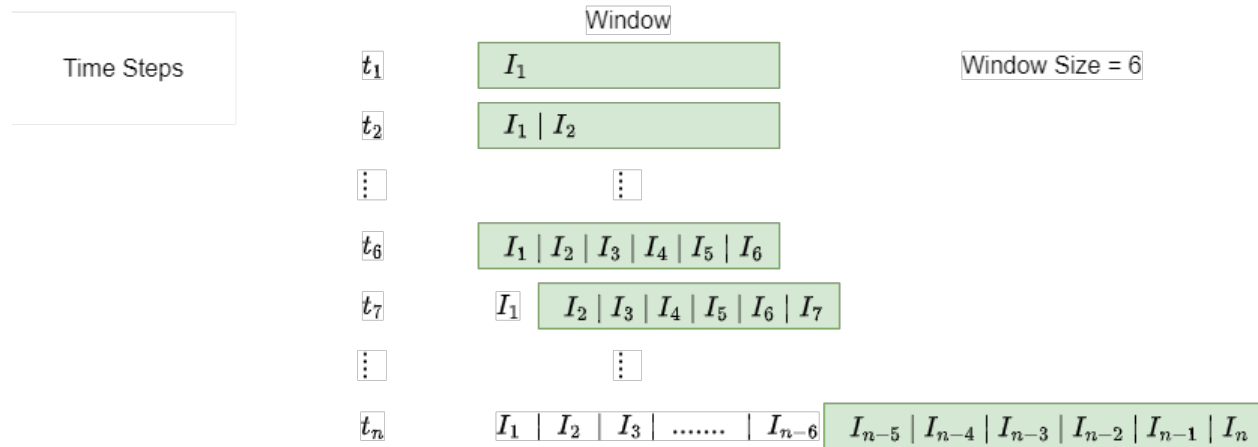
### Cross Reference

- [Howto BF-STREAMS-001: Accessing Native Data From MLPro](#)
- [API Reference: Streams](#)

## Stream Tasks

### Window

In streaming scenarios, data is available sequentially, and the amount of data received is directly proportional to the time for which the stream is active. In practice, this data accumulates in tremendous amounts as the application becomes complex. Processing data with minimum use of storage is important. A window task stores a small amount of data from the incoming stream, that can be used to process subsequent tasks based on a smaller amount of data that represents the stream behaviour.



The window task in MLPro, stores the most recent instances received from the stream in the buffer. The buffer size of the window is fixed and defined by the user. As soon as the buffer is full, the oldest instance is deleted from the buffer to add the latest instance to the buffer. The subsequent tasks in the workflow with dependency on window, have access to the data in the buffer.

**Note:** The availability of the buffered instances to the subsequent tasks can be delayed by setting the `p_delay` parameter to `True`. In this case, the buffered instances are only available once the buffer is completely full.

The window task of MLPro, also provides functionality to get statistical information about the buffered instances, such as Boundaries, Mean, Variance and Standard Deviation of the features of the instances in the buffer. Additionally, MLPro also provides visualization functionality for window, as shown below.

### Cross Reference

- [Howto BF-STREAMS-110: Window](#)
- [API Reference: Streams](#)

## Rearranger

A stream object consists of a Feature Space with a number of features defining each instance at a given timestep. A Rearranger task in MLPro maps the features of an input stream instances to a user defined feature space as an output. In other words, a rearranger task can be used to filter out un-interested features of a stream object for a particular task.

### Cross Reference

- [Howto BF-STREAMS-111: Rearranger \(2D\)](#)
- [Howto BF-STREAMS-111: Rearranger \(3D\)](#)
- [Howto BF-STREAMS-111: Rearranger \(nD\)](#)
- [API Reference: Streams](#)

## Deriver

The deriver task in MLPro provides a functionality to derive a selected feature and extend the feature with its derivative. In mathematics, the derivative is a concept that measures the rate of change of a function at a particular point. It is represented as the slope of the tangent line to a function at that point. The derivative is a fundamental tool in calculus and is used to study the properties of functions, such as their maxima, minima, and inflection points. It is also used in various scientific and engineering fields to study the behavior of systems that change over time, such as in the modeling of physical processes and in the analysis of dynamic systems. The derivative can be calculated using limits or through a number of differentiation rules for common functions, such as power, exponential, and logarithmic functions.

In the current implementation, we set up the basic formula of the derivation, as follows:

$$f'(x) = \lim_{h \rightarrow 0} (f(x+h) - f(x)) / h$$

---

**Note:** The order of derivative can be selected through `p_order_derivative`. If you would like to have two orders of derivative, then you have to add two separate tasks to the workflow.

---

### Cross Reference

- [Howto BF-STREAMS-114: Deriver](#)
- [API Reference: Streams](#)

## 3rd Party Support

As part of the MLPro project, numerous public data providers have been and are being integrated. Their data sets are mapped to MLPro's stream concept using wrapper technology and can, therefore, be easily consumed in your own ML applications.

Examples to be mentioned here are:

- [MLPro-Int-OpenML - Integration of OpenML into MLPro](#)
- [MLPro-Int-scikit-learn - Integration of scikit-learn into MLPro](#)
- [MLPro-Int-River - Integration of River into MLPro](#)

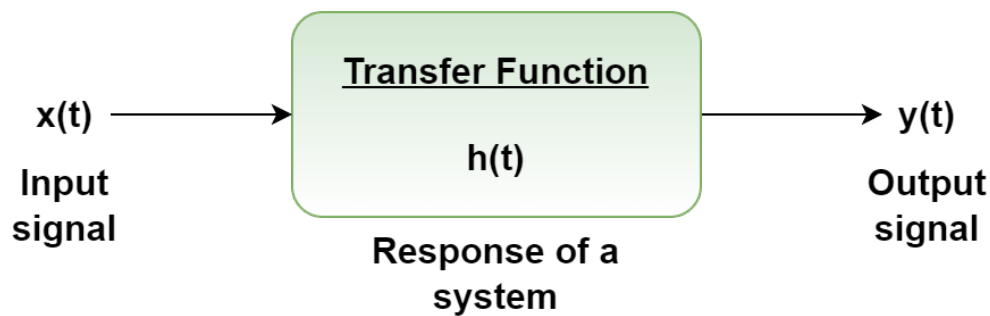
A complete list of all approved MLPro extensions can be found in the MLPro Extension Hub.

### 3.5.2 Physics

BF-PHYSICS is part of the third layer of MLPro-BF for application support, which provides elementary functionalities for the following topics:

#### Transfer Function

A transfer function is a mathematical representation of the relationship between the input and output of a time-invariant system. It is commonly used in control theory and electrical engineering to analyze and design systems with inputs and outputs, but it is not restricted only to those aspects. The transfer function provides valuable functionality to process inputs to outputs within a period of time. It can be used to design controllers that regulate the behaviour of the system, predict its response to inputs, and analyze the performance of the system in the frequency domain. Transfer functions play an important role in the design and analysis of control systems, communication systems, and signal processing systems.



In MLPro, there are three possibilities for transfer functions, which are:

1. Linear function
2. Custom function
3. Function approximation (future work)

Transfer Function class can be accessed as follows:

```
from mlpro.bf.physics import TransferFunction
```

#### Cross Reference

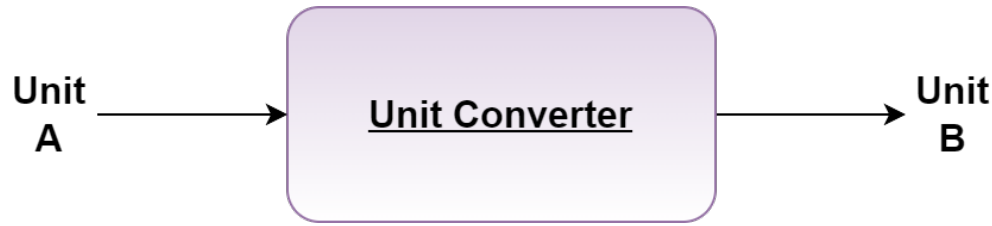
- [Howto BF-PHYSICS-001: Transfer Functions](#)
- [Howto BF-PHYSICS-002: Unit Converter](#)
- [API Reference](#)

#### Unit Converter

MLPro's unit converter is a functionality to convert common units of measurements, such as:

- Length
- Pressure
- Electric Current
- Time
- Temperature

- Mass
- Power
- Force



Unit Converter class can be accessed as follows:

```
from mlpro.bf.physics.unitconverter import UnitConverter
```

#### Cross Reference

- [Howto BF PHYSICS 002](#)
- [API Reference](#)

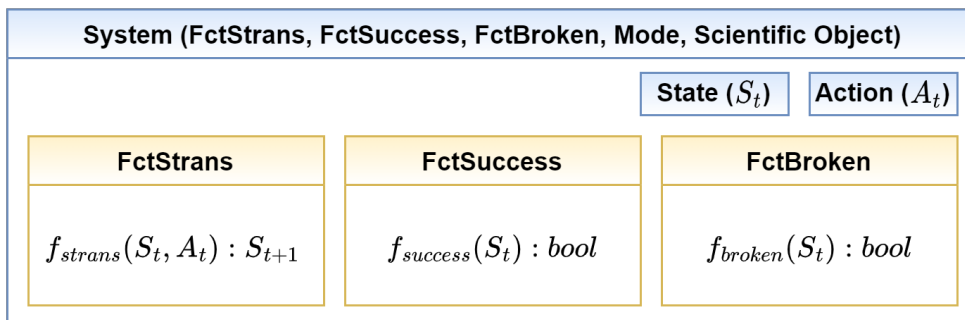
Physics-related functionality in a framework typically includes modules for dealing with simulating physical systems, performing numerical calculations and analysis, and many more.

### 3.5.3 State-based Systems

MLPro aims to standardize machine learning processes to accommodate complex applications in simplified reusable APIs. MLPro's Systems module standardizes state-based systems and their operation in a modular design. The keyword **state-based** implies the possibility to characteristically represent a system's unique state as a vector corresponding to a given timestep.

A state-based system has a definite condition at any given point in time, defined by a fixed number of variables that completely defines the condition. A system transits from a state to next state at each timestep based on the inherent state transition dynamics. However, this state transition is triggered by an external source of action, for example an Actuator.

In real application of state-based systems, such as controlled systems, it is highly interesting to maintain a desired system state, reach a system state or maximize system output, through optimum state transitions. Additionally, it is also an important concern to verify if the system is performing within the objective of the application or if the system has failed.





As shown in the figure above, MLPro's Systems module encapsulates the aforementioned functionalities into a standard template. The System object of MLPro can be reused to define any custom system with default methods to handle surrounding standard operations.

The system's module provides following objects and templates:

#### 1. **System:**

The System class standardizes and provides the base template for any State-based System along with standard ML-Pro functionalities such as Logging, Timer, Cycle Management, Persistence, Real/Simulated mode and Reset. The System class additionally provides room for custom functionalities such as Reaction Simulation, Terminal State Monitoring such as Success and Broken. These custom functionalities can be incorporated by implementing the `_simulate_reaction()`, `_compute_success()`, `_compute_broken()` methods on Systems class or corresponding function classes (described below), which are then passed as a parameter to the system.

---

**Note:** The System class also supports operation in modes: **Real** and **Simulated**, based on which, it enables working with a real hardware or a simulated system respectively.

---

#### 2. **FctStrans:**

The FctStrans (State Transition Function) standardizes the process of simulating the primary State Transition process of a System. The `simulate_reaction(p_state, p_action)` method of this class takes the current state of the environment and the action from the corresponding actuator as a parameter, and maps it to the next state of the system, based on the inherent dynamics.

---

**Note:** Please implement the `_simulate_reaction()` method of FctStrans, in order to re-use in a custom implementation.

---

#### 3. **FctSuccess:**

A System state can be monitored through FctSuccess (Success Function) to determine if the system has reached the expected objective state/output. It maps the current state of the system to a boolean value indicating the success of a system.

---

**Note:** Please implement `_compute_success()` method of FctSuccess, in order to re-use it in a custom implementation.

---

#### 4. **FctBroken:**

Similar to FctSuccess class, the FctBroken class standardizes the process of monitoring whether the system has reached a broken terminal state, by mapping the current state to a boolean value indicating the broken state.

---

**Note:** Please implement `_compute_broken()` method of FctBroken, in order to re-use it in custom implementation.

---

#### 5. **State:**

The state object represents the current state of the system with respect to time. A state object inherits from the Element class of MLPro, which represents an element in a Multi-dimensional Set object, a State-Space in this case. The state consists information about the System for corresponding dimension of the related State-Space.

#### 6. **Action:**

The Action object standardizes external input to the system. For example, input from a controller, input from an actuator or an agent in case of Reinforcement Learning. The standard Action object consists of an ActionElement or a list of

ActionElements, in case of more than one action sources. The action element is similar to a state object, consisting corresponding values for all the dimension in the related action-space.

MLPro also provides the possibility to integrate real world hardware, such as controllers and hardware to the System object. Furthermore, Systems module integrates optional visualization and simulation functionalities from MuJoCo into MLPro for re-usability.

#### 7. DemoScenario:

Our module provides a demo scenario class to help users validate their system's behavior. The Demo scenario takes a system as a parameter and runs a particular scenario to validate its behavior. Currently, this scenario does not consider an external object to compute and deliver the actions. Rather, it computes the action internally and processes the action in the system. The demo scenario supports two action styles:

- **Constant Action:**

A constant value of action is given for the entire run.

- **Action List:**

A list of actions provided by the user is used to process the system for a given number of cycles. With this demo scenario class, users can easily test their system and ensure that it behaves as expected.

**Learn more**

## MuJoCo Integration

MuJoCo is a well-known physics engine for its fast and accurate simulation. The aim is to facilitate research and development in robotics, biomechanics, graphics and animation, and other areas. More explanation about MuJoCo can be found in [here](#).

In order to use the MuJoCo integration in MLPro, the following steps need to be done:

- **Create a MuJoCo Model**

Create a MuJoCo model file accordingly to your design. Some example model are published by MuJoCo and can be accessed [here](#). Below is an example of MuJoCo model file.

```
<mujoco>
  <option timestep="0.05" gravity="0 0 -9.81" integrator="RK4">
    <flag sensornoise="enable" energy="enable"/>
  </option>
  <worldbody>
    <light diffuse=".5 .5 .5" pos="0 0 3" dir="0 0 -1"/>
    <geom type="plane" size="1 1 0.1" rgba=".9 0 0 1"/>

    <body name="link1" pos="0 0 2" euler="0 0 0">
      <joint name="pin" type="hinge" axis = "0 -1 0" pos="0 0 0.5"/>
      <inertial pos="0 0 0" mass="1" diaginertia="1 1 1" />
      <geom type="cylinder" size="0.05 0.5" rgba="0 .9 0 1"/>
    </body>

  </worldbody>

  <actuator>
    <motor joint="pin" name="torque1" gear="1" ctrllimited="true" ctrlrange=
    ↪ "-50 50"/>
  </actuator>
</mujoco>
```

(continues on next page)

(continued from previous page)

```
</actuator>
</mujoco>
```

The above model simulates one body called `link1` which has a cylindrical shape. It is attached to the world body with a joint called `pin` with the type of hinge. This joint is controlled by an actuator called `torque1` boundaries between -50 and 50.

- **Create a System**

When you instantiate the System, put the MuJoCo model file path on `p_mujoco_file`. If the model is correct and the path is correct, then the wrapper will automatically wrap the state and action space based on the MuJoCo model.

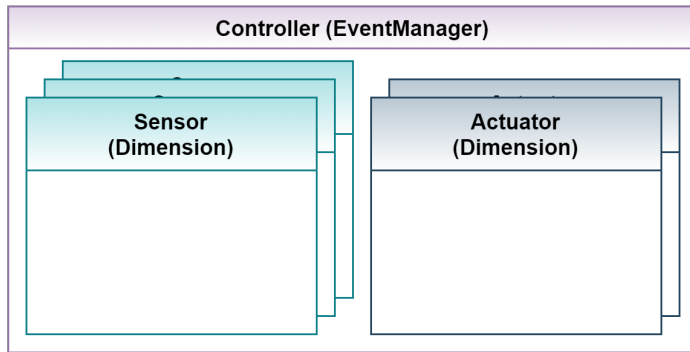
If you want to view your model only before including it in MLPro, you can use the MuJoCo tool by dragging and dropping the model file into it. The tool can be downloaded [here](#).

### Cross Reference

- [MuJoCo Tool](#)
- [MuJoCo XML Reference](#)
- [MuJoCo Model Samples](#)
- [Unity Plug-in for MuJoCo](#)
- [MuJoCo Wrapper](#)
- [Howto BF-SYSTEMS-002: Double Pendulum Systems wrapped with MuJoCo](#)
- [Howto BF-SYSTEMS-003: Cartpole Continuous Systems wrapped with MuJoCo](#)
- [Howto BF-SYSTEMS-004: MuJoCo Simulation with Camera](#)
- [Howto RL-AGENT-021: Train and Reload Single Agent Cartpole Discrete \(MuJoCo\)](#)
- [Howto RL-AGENT-022: Train and Reload Single Agent Cartpole Continuous \(MuJoCo\)](#)
- [Howto RL-ATT-002: Train and Reload Single Agent using Stagnation Detection Cartpole Discrete \(MuJoCo\)](#)
- [Howto RL-ATT-003: Train and Reload Single Agent using Stagnation Detection Cartpole Continuous \(MuJoCo\)](#)
- [Howto RL-ENV-005: Run Agent with random policy on double pendulum mujoco environment](#)

### Hardware Access

MLPro also provides possibility to use the standardized Systems API with real world systems. It manages the interaction between real world hardware including sensors, actuators and controllers with the standard processes in MLPro framework.



As shown in the above figure, the controller class in MLPro registers a number of sensors and actuators for a system.

1. **Sensor:**

A sensor observes a system to deliver characteristic information about the system at a given time. The Sensor class in MLPro inherits from the Dimension class.

2. **Actuator:**

An actuator is responsible to generate an Action, which is executed in the real world system. Similar to Sensor class, the Actuator class is also inherited from the Dimension class of MLPro.

3. **Controller:**

The controller is responsible to gather sensor data, compute the error signal and generate a corresponding action in order to maintain/reach the desired state of the system. The Controller class in MLPro manages the mapping details to map actions and states, to and from Actuators and Sensors, respectively.

### Cross Reference

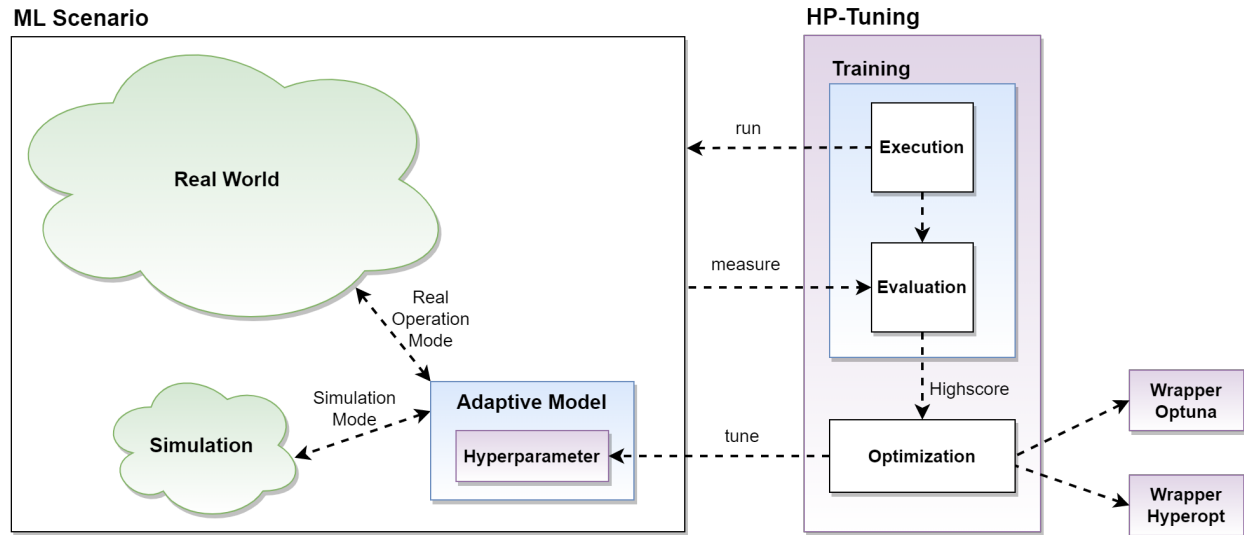
- *Howto BF-SYSTEMS-001: System, Controller, Actuator, Sensor*
- *API Reference BF-Systems*

### Cross Reference

- *Howto BF-SYSTEMS-001: Demonstrating Native Systems*
- *Howto BF-SYSTEMS-010: System, Controller, Actuator, Sensor*
- *Howto BF-SYSTEMS-011: Systems wrapped with MuJoCo*
- *Howto BF-SYSTEMS-012: Cartpole Continuous Systems wrapped with MuJoCo*
- *Howto BF-SYSTEMS-013: MuJoCo Simulation with Camera*
- *API Reference BF-Systems*
- *API Reference BF-Systems Sample Pool*

## 3.6 Layer 4 - Machine Learning

One of the fundamental concepts in MLPro is to anchor universal standards for machine learning already in the basic functions. This shall facilitate the creation of higher ML functionalities and ensure their recombability. The challenge here is to capture the nature of machine learning on an abstract and general level while establishing concrete templates and processes and solving elementary subtasks. The highest layer 4 of the basic functions of MLPro is dedicated to this topic.



The focus of the consideration is the *adaptive model* with its elementary properties

- Adaptivity
- Executability
- Parameterizability
- Persistence

Of course, this model does not exist just on its own. Rather, it interacts with a simulated or real object. For example, in the case of offline supervised learning, this can be a data set, in the case of online unsupervised learning, a data stream, or in the case of reinforcement learning, a state-based system. Topics like this are covered in MLPro in higher-level ML frameworks. On an abstract level, however, we introduce the *ML scenario* for this because although we do not yet know anything about the concrete ML application, we know that an adaptive model is involved. Furthermore, we propagate that the application is executable in “simulation” or “real operation” mode.

In MLPro, a model’s *training and the tuning* of its hyperparameters are based on such an ML scenario. For tuning, powerful packages from third parties are already integrated at this low level using wrapper technology.

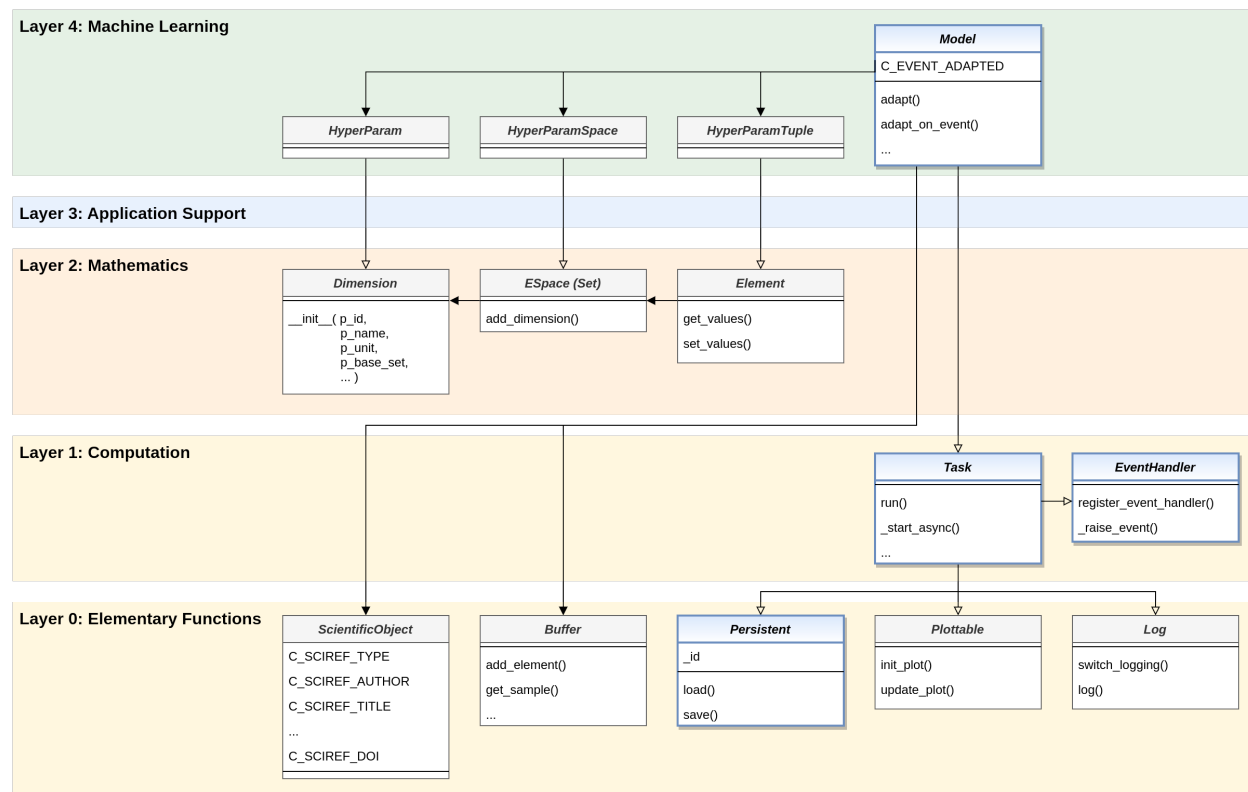
In more complex applications, it can be helpful to group multiple models and allow communication between them. It would be desirable here to optimally utilize the system resources through parallel processing. For this purpose, MLPro provides *adaptive workflows*.

In the field of systems engineering, the creation of suitable simulations or digital twins is an essential aspect. However, suppose a system cannot be described mathematically precisely enough due to its complexity or unknown influencing factors. In that case, machine learning methods can also be used to imitate the system behavior based on historical data and/or online monitoring. For this purpose, MLPro provides standards for adaptive systems.

**Learn more**

### 3.6.1 The Adaptive Model

MLPro provides the central template class **Model** for adaptive models. This bundles all properties important for machine learning on an abstract level. It represents the basis for all higher adaptive classes of the entire MLPro ecosystem and inherits its essential properties and possibilities for application-specific adjustments to them.



#### Performant Execution

As shown in the simplified class diagram above, the **Model** class is made up of numerous base classes of the lower levels through inheritance. So, from *Layer 1 - Computation*, it inherits the executability and asynchronous processing capabilities of class **Task**. In this way, it can also be combined in workflows to (parallelly/asynchronously) executable groups of models. From class **EventHandler** of the same level, it inherits the ability to raise events and forward them to registered event handlers.

#### Persistence

In particular, from *Layer 0 - Elementary Functions*, it inherits the ability of the **LoadSave** class to be able to be saved and reloaded. Other elementary capabilities such as logging, visualization, buffering of sample data, and referencing a scientific source are also fed in from this lower level.

#### Adaptivity

The **Model** class itself adds the ability to adapt. To this end, two mechanisms are introduced that support **explicit adaptation** based on external data and **event-oriented adaptation**. In both cases, the event **C\_EVENT\_ADAPTED** is raised, which can also be optionally handled as part of event handling. In this way, adaptation cascades can be triggered in a group of cooperating models.

#### Hyperparameters

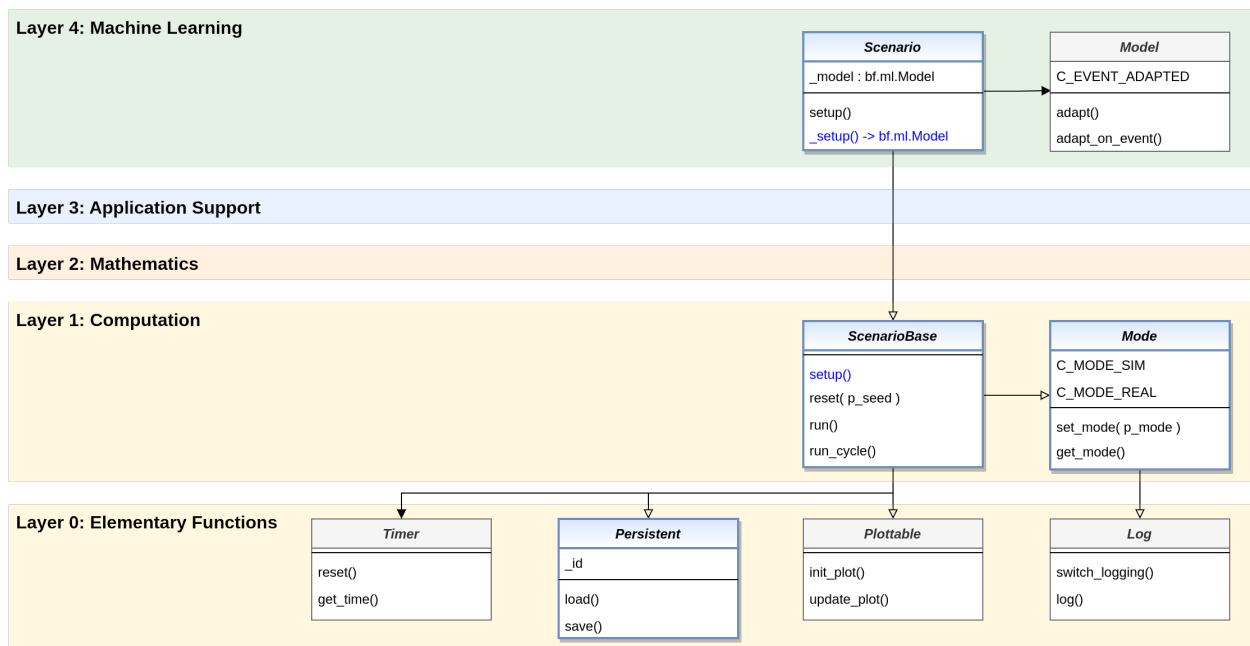
Also, at the top layer 4 for machine learning, a system for hyperparameters is introduced and added to the **Model** class. These, in turn, take up the concepts of **Dimension**, **Set/Space**, and **Element** from *Layer 2 - Mathematics*.

#### Cross Reference

- *Howto BF-ML-001: Adaptive Model*
- *Howto BF-ML-010: Hyperparameters*
- *Models in Reinforcement Learning: Policy, Agent, MultiAgent*
- *Models in Game Theory: Player, MultiPlayer*
- *BF-Events - Event Handling*
- *API Reference BF-ML*

### 3.6.2 ML Scenarios

As already mentioned, adaptive models in MLPro are combined with their concrete context to form an ML scenario. MLPro provides the abstract template class **bf.ml.Scenario** for this. At this level, this is not yet intended for use in your own customer applications, but is only used here to standardize the basic properties of an ML scenario.



From the root class of all scenarios in MLPro *bf.ops.ScenarioBase* it inherits the following properties

- Operation mode (simulation or real operation)
- Execution of cycles
- Persistence
- Visualization

and adds at this level the management of an internal adaptive model.

#### Cross Reference

- *Class bf.ops.ScenarioBase*
- *API Reference BF-ML*

### 3.6.3 Training and Tuning

A template for training models in their defined context is also introduced at this level. In a broader sense, this also includes finding an optimal value assignment for their hyperparameters. In MLPro, the **Training** class defines standards for this. Although abstract at this level, it fully implements the hyperparameter tuning here. The basic concept pursued here envisages executing an ML scenario under defined conditions and allowing the model contained therein to learn.

#### Persistence of Training Results

At the end of the training, the training results are saved in the file system. In particular, the entire scenario is saved here for later operational use. This includes both the trained model and the context in the last state.

#### Scoring

One of the training results is the **highscore**. Its determination is of course heavily dependent on the type of learning and can therefore only be specified in higher layers of MLPro. In any case, however, it is basically a real number that allows a qualitative statement about the learning performance of the model in its scenario.

#### Hyperparameter Tuning

Hyperparameter tuning is an optional training function performed by its own **HyperParamTuner** class. In particular, it defines the **maximize** method, which maximizes the highscore of a designated training by varying the hyperparameters of the model it contains. The optimization itself is not performed natively by MLPro, but by third-party packages. To this purpose, MLPro provides wrappers for Optuna and Hyperopt.

#### Cross Reference

- [API Reference BF-ML](#)
- Wrapper for Optuna
- Wrapper for Hyperopt

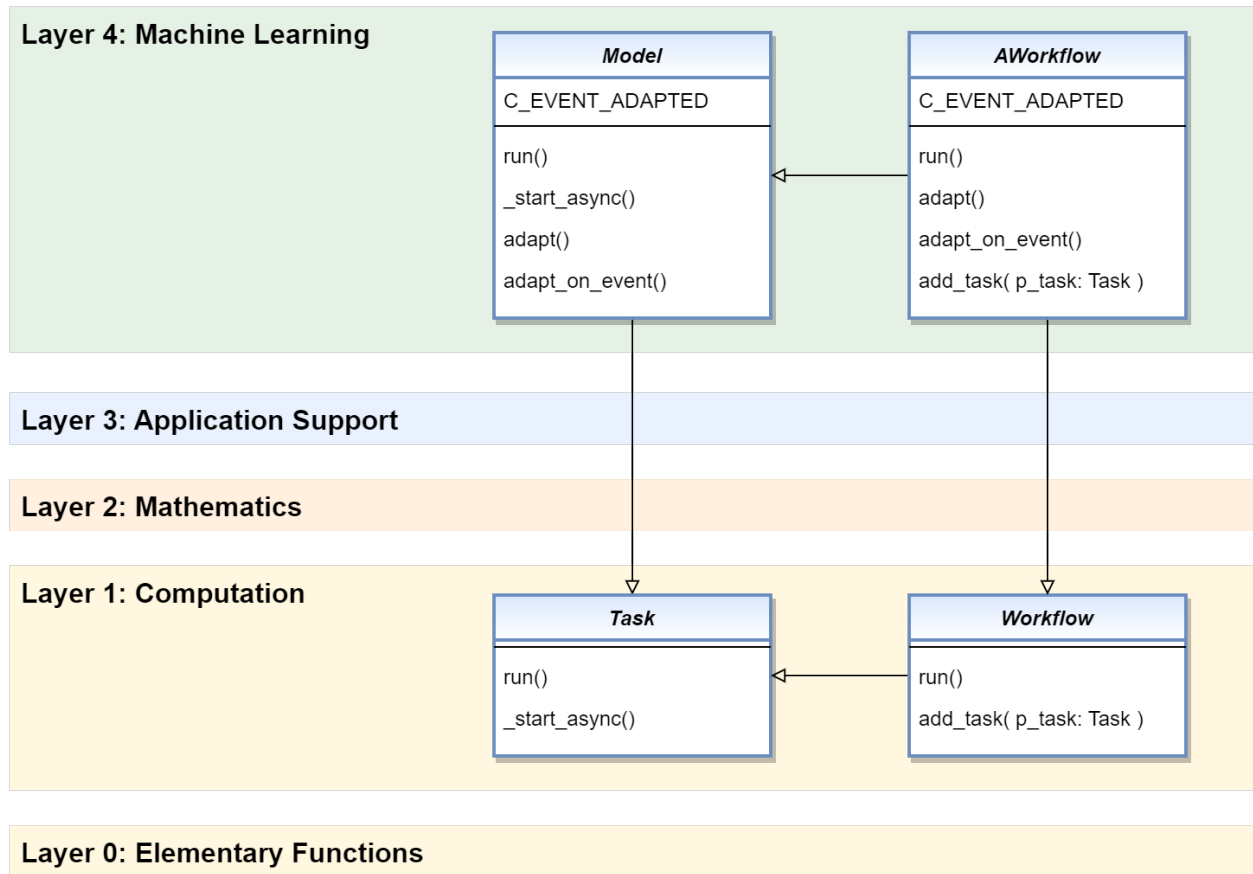
### 3.6.4 Adaptive Workflows

The **Model** class inherits from the **bf.mt.Task** class, among others. In combination with another class **AWorkflow**, all possibilities of MLPro's multitasking abilities are unlocked for machine learning:

1. Models can execute internal methods asynchronously
2. Models can run as separate threads or processes
3. Models can be grouped into macro models in workflows
4. Models can share/exchange data using a shared object

This enables the efficient execution and adaptation of models and model groups using all available runtime resources.

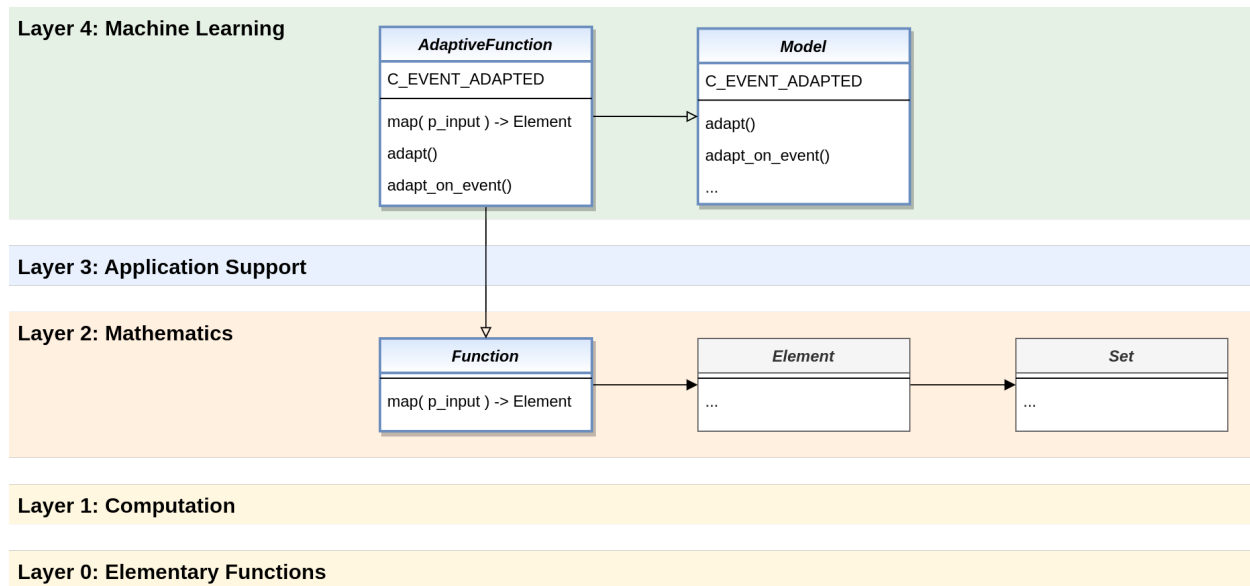


**Cross Reference**

- *BF-MT: Multitasking*

**3.6.5 Adaptive Functions**

A special kind of adaptive models are the **adaptive functions**. They combine the properties of a mathematical function with those of an adaptive model. The **class AdaptiveFunction** manifests and standardizes these functions without implementing concrete learning paradigms. The latter happens in higher MLPro frameworks, for example in MLPro-SL for offline/online supervised learning.



Adaptive functions have a high practical relevance. They are reused within MLPro, e.g. in connection with model-based agents. But they can also be used in general for predictions.

#### Cross Reference

- *BF-Math: Mathematics*
- *SL: Adaptive Functions for Supervised Learning*
- *RL: Model-based Agents*

### 3.6.6 Adaptive Systems (coming soon)

This functionality is in preparation. Further explanation coming soon...

#### Cross Reference

- *BF-Systems: State-based Systems*

#### Cross Reference

- *Related Howtos*
- *API Reference BF-ML - Machine Learning*
- *API Reference BF-ML-Systems - Adaptive Systems*

## MLPRO-SL - SUPERVISED LEARNING

### 4.1 Overview

MLPro provides a subtopic package for supervised learning, namely MLPro-SL. At the moment, the implementation is still limited but we are working on it and improving it to bring you full supervised learning functionalities in the near future. MLPro-SL is designed to handle online and offline supervised learning, which means that the model can be used for different purposes, e.g. model-based reinforcement learning, online adaptivity, and more.

The current implementation covers:

- A base class of an adaptive function for supervised learning
- A base class of an adaptive function for feedforward neural networks, including MLP
- Ready-to-use PyTorch-based MLP networks in the pool of objects

#### Learn more

- *Getting started with MLPro-SL*

#### Cross Reference

- *API Reference: MLPro-SL*
- *API Reference: MLPro-SL Pool of Objects*

### 4.2 Getting Started

As mentioned in the introductory section, MLPro-SL's functionalities are still limited and not ready to be labelled as the first version. However, we are working on it to enhance MLPro-SL and bring you full supervised learning functionalities soon.

At the moment, we provide a basic template class for *supervised learning adaptive function*, which has been extended to the feedforward neural network. We introduce MLP as a sample of the MLPro-SL model and provide a ready-to-use PyTorch-based multilayer perceptron network.

After following the below step-by-step guideline, we expect the user understands the MLPro-SL in practice and starts using MLPro-SL.

#### 1. What is MLPro?

If you are a first-time user of MLPro, you might wonder what is MLPro. Therefore, we recommend initially start with understanding MLPro by checking out the following steps:

- (a) *MLPro: An Introduction*
- (b) *introduction video of MLPro*

- (c) [\*installing and getting started with MLPro\*](#)
- (d) [\*MLPro paper in Software Impact journal\*](#)

## 2. What is Supervised Learning?

If you have not dealt with supervised learning, we recommend starting to understand at least the basic concept of supervised learning. There are plenty of references, articles, papers, books, or videos on the internet that explains supervised learning. As an overview, supervised learning is a type of machine learning in which a model is trained on a labelled dataset to predict the output for new/unseen inputs. Supervised learning can be used to build a predictive model that can make predictions based on available data.

## 3. What is MLPro-SL?

We expect that you have a basic knowledge of MLPro and supervised learning. Therefore, you need to understand the overview of MLPro-SL by following the steps below:

- (a) [\*MLPro-SL introduction page\*](#)

## 4. Understanding Adaptive Function in MLPro-SL

First of all, it is important to understand the adaptive function in MLPro-SL, which can be found on [\*this page\*](#).

Then, you can start following some of our howto files related to the adaptive function in MLPro-SL, which is used for model-based RL, as follows:

- (a) [\*Howto RL-MB-001: Train and Reload Model Based Agent \(Gym\)\*](#)
- (b) [\*Howto RL-MB-002: MBRL with MPC on Grid World Environment\*](#)

For more advanced supervised learning technique in model-based RL, e.g. applying a native model-based RL network, here is an example that can be used as a reference:

- (c) [\*Howto RL-MB-003: MBRL on RobotHTM Environment\*](#)

## 5. Additional Guidance

After following the previous steps, we hope that you could practice MLPro-SL and start using this subpackage for your SL-related activities. For more advanced features, we highly recommend you to check out the following files:

- (a) [\*API Reference: MLPro-SL\*](#)
- (b) [\*API Reference: MLPro-SL Pool of Objects\*](#)

## 4.3 Adaptive Functions

In supervised learning, the adaptive function refers to the ability of the model to adjust its parameters in response to new input data. Specifically, it refers to the ability of the model to learn from the labelled training data and improve its performance on new/unseen data. During the training phase, the model is presented with a set of input features and the corresponding output labels, and it adjusts its parameters (e.g. weights and biases) to minimize the error between its predicted outputs and the true labels. This process of updating the model's parameters is often referred to as adaptation. The goal of this learning process is for the model to be able to accurately predict the output for input data. This is known as the model's generalization performance, and it is a key measure of its adaptive function.

We provide ready-to-use adaptive function models in MLPro-SL's pool of objects, which can be found, as follows:

### 4.3.1 Adaptive Function Pool

#### Adaptive Function Pytorch

First of all, it is important to understand the adaptive function in MLPro-SL, which can be found on [this page](#).

In this functionality, we integrate PyTorch into MLPro-SL, thus PyTorch functionalities can be reused in MLPro-SL.

At the moment, we provide only PyTorch-based multilayer perceptron in the pool, but recurrent neural networks and transformers are planned to be included in the next version shortly.

#### Cross Reference

- *Howto RL-MB-001: Train and Reload Model Based Agent (Gym)*
- *Howto RL-MB-002: MBRL with MPC on Grid World Environment*
- *API Reference: MLPro-SL Pool of Objects*

#### Cross Reference

- *BF-ML: Adaptive Functions*
- *API Reference: MLPro-SL*



## MLPRO-OA - ONLINE ADAPTIVITY

This framework addresses topics of online machine learning and is currently in progress.

### 5.1 Overview

Further descriptions coming soon...

### 5.2 Getting Started

Further descriptions coming soon...

### 5.3 Online Adaptive Stream Processing

Further descriptions coming soon...

### 5.4 Online Adaptive Systems

Further descriptions coming soon...





## MLPRO-RL - REINFORCEMENT LEARNING

### 6.1 Overview

MLPro-RL is the first ready-to-use subpackage in MLPro that is intended for reinforcement learning (RL)-related activities. MLPro-RL provides complete base classes of the main RL components, e.g. agent, environment, policy, multiagent, and training. The training loop is developed based on the Markov Decision Process (MDP) model, as shown in the following diagram.

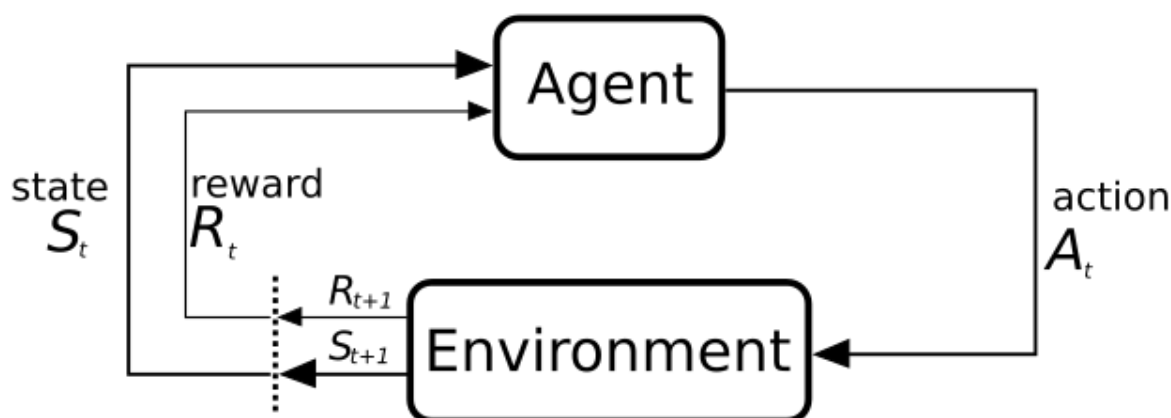


Fig. 1: This figure is taken from [Sutton and Barto](#), licensed by CC BY-NC-ND 2.0.

An MDP model contains two major components, such as the environment and the agent. The agent can be considered as the decision maker, who chooses actions based on its policy by taking into account the current state of the environment. The environment is the surrounding where the agent lives and interacts. The actual condition in the environment is represented by states. MDP formulates the interaction between the agent and the environment, where the agent selects an action and sends the action to the environment. The environment reacts to the given action that makes the condition in the environment change. Then, the environment sends back the information in the form of states and reward to indicate the actual condition in the environment and the impact of the taken action on the environment respectively. Afterwards, the agent can adapt its policy and repeat the interactions until reaching optimality.

MLPro-RL can handle a broad scope of RL training, including model-free RL or model-based RL, single-agent or multi-agent, and simulation or real hardware mode. Hence, this subpackage can be a one-stop solution for students, educators, RL engineers or RL researchers to support their RL-related tasks. The structure of MLPro-RL can be found in the following figure.

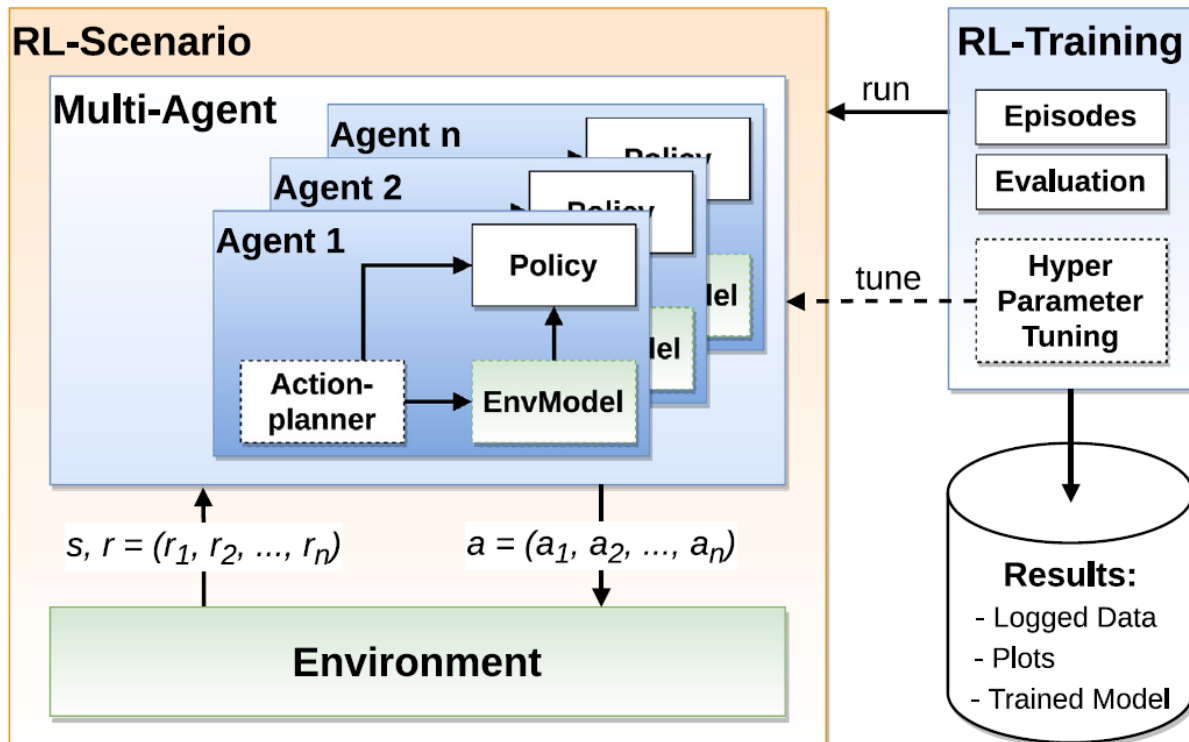


Fig. 2: This figure is taken from [MLPro 1.0 paper](#).

If you are interested to utilize MLPro-RL, you can easily access the RL modules, as follows:

```
from mlpro.rl import *
```

Additionally, you can find the more comprehensive explanations of MLPro-RL including a sample application on controlling a UR5 Robot in this paper: [MLPro 1.0 - Standardized Reinforcement Learning and Game Theory in Python](#).

### Learn more

- [Getting started with MLPro-RL](#)

### Cross Reference

- [Related Howtos](#)
- [API Reference: MLPro-RL](#)
- [API Reference: MLPro-RL Pool of Objects](#)
- [MLPro 1.0 Paper](#)
- [MLPro GitHub](#)

## 6.2 Getting Started

Here is a concise series to introduce all users to the MLPro-RL in a practical way, whether you are a first-timer or an experienced MLPro user.

If you are a first-timer, then you can begin with **Section (1) What is MLPro?**.

If you have understood MLPro but not reinforcement learning, then you can jump to **Section (2) What is Reinforcement Learning?**.

If you have experience in both MLPro and reinforcement learning, then you can directly start with **Section (3) What is MLPro-RL?**.

After following the below step-by-step guideline, we expect the user understands the MLPro-RL in practice and starts using MLPro-RL.

### 1. What is MLPro?

If you are a first-time user of MLPro, you might wonder what is MLPro. Therefore, we recommend initially start with understanding MLPro by checking out the following steps:

- (a) [MLPro: An Introduction](#)
- (b) [introduction video of MLPro](#)
- (c) [installing and getting started with MLPro](#)
- (d) [MLPro paper in Software Impact journal](#)

### 2. What is Reinforcement Learning?

If you have not dealt with reinforcement learning, we recommend starting to understand at least the basic concept of reinforcement learning. There are plenty of references, articles, papers, books, or videos on the internet that explains reinforcement learning. But, for deep understanding, we recommend you to read the book from Sutton and Barto, which is [Reinforcement Learning: An Introduction](#).

### 3. What is MLPro-RL?

We expect that you have a basic knowledge of MLPro and reinforcement learning. Therefore, you need to understand the overview of MLPro-RL by following the steps below:

- (a) [MLPro-RL introduction page](#)
- (b) [Section 4 of MLPro 1.0 paper](#)

### 4. Understanding Environment in MLPro-RL

First of all, it is important to understand the structure of an environment in MLPro, which can be found on [this page](#).

Then, you can start following some of our howto files related to the environment in MLPro-RL, as follows:

- (a) [Howto RL-001: Reward](#)
- (b) [Howto RL-AGENT-001: Run an Agent with Own Policy](#)

### 5. Understanding Agent in MLPro-RL

In reinforcement learning, we have two types of agents, such as a single-agent RL or a multi-agent RL. Both of the types are covered by MLPro-RL. To understand the different possibilities of an agent in MLPro, you can visit [this page](#).

Then, you need to understand how to set up a single-agent and a multi-agent RL in MLPro-RL by following these examples:

- (a) [Howto RL-AGENT-001: Run an Agent with Own Policy](#)
- (b) [Howto RL-AGENT-003: Run Multi-Agent with Own Policy](#)

## 6. Selecting between Model-Free and Model-Based RL

In this section, you need to select your direction of the RL training, whether it is a model-free RL or a model-based RL. However, firstly, you can pay attention to these two pages, which are [RL scenario](#) and [training](#), before selecting either of the paths below.

- Model-Free Reinforcement Learning

To practice model-free RL in the MLPro-RL package, here are a video and some ready-to-use howto files that can be followed:

- (a) [A sample application video of MLPro-RL on a UR5 robot](#)
- (b) Howto RL-AGENT-002: Train an Agent with Own Policy
- (c) Howto RL-AGENT-004: Train Multi-Agent with Own Policy

- Model-Based Reinforcement Learning

Model-based RL contains two learning paradigms, such as learning the environment (model-based learning) and utilizing the model (e.g. as an action planner). To practice model-based RL in the MLPro-RL package, here are a howto file that can be followed:

- (a) [Howto RL-MB-001: Train and Reload Model Based Agent \(Gym\)](#)
- (b) Howto RL-MB-002: MBRL with MPC on Grid World Environment

For more advanced MBRL technique, e.g. applying a native MBRL network, here is an example that can be used as a reference:

- (c) Howto RL-MB-003: MBRL on RobotHTM Environment

## 7. Additional Guidance

After following the previous steps, we hope that you could practice MLPro-RL and start using this subpackage for your RL-related activities. For more advanced features, we highly recommend you to check out the following howto files:

- (a) Howto RL-AGENT-011: Train and Reload Single Agent (Gym)
- (b) Howto RL-AGENT-021: Train and Reload Single Agent (MuJoCo)
- (c) Howto RL-HT-001: Hyperopt
- (d) Howto RL-HT-002: Optuna
- (e) Howto RL-ATT-001: Stagnation Detection
- (f) Howto RL-ATT-002: SB3 Policy with Stagnation Detection

## 6.3 Environments

In RL, the environment refers to the physical, virtual, or abstract system in which the agent interacts and learns. The environment is the source of stimuli that the agent perceives and the arena in which it takes actions.

The environment is defined by a set of states, actions, and transition dynamics. The state space is the set of all possible states that the agent can observe, and the action space is the set of all possible actions that the agent can take. The transition dynamics describe how the environment changes in response to the agent's actions.

The environment also provides the agent with a reward signal that indicates how well it is doing in terms of achieving its goals. The reward function is a mapping from states and actions to real-valued scalars that quantifies the desirability of each state-action pair.

The agent interacts with the environment over a sequence of time steps. At each time step, the agent observes the current state of the environment and selects an action. The environment then transitions to a new state and returns a reward signal to the agent.

Overall, the environment in RL provides the agent with the necessary information to learn a policy that maps states to actions and maximizes the cumulative reward signal. The environment can be real-world or simulated, and can be described by a mathematical model or a black box.

MLPro-RL supplies two main classes for an environment to support model-free and model-based RL. The first base class is `Environment`, which has a role as a template for designing environments for both approaches. The second base class is `EnvModel`, which is adaptive and utilized in model-based RL. Both `Environment` and `EnvModel` classes inherit a common base class `EnvBase` and its fundamental properties, e.g. state and action space definition, reset the corresponding environment method, state transition method, etc.

There are two main possibilities to set up an environment in MLPro, such as,



### 6.3.1 Developing Custom Environments

#### MLPro Environment Model

- Latency defines the time needed for the environment to react to the input.
- Supports single and multi-agent control (see C\_REWARD\_TYPE).
- Supports simulation and real control mode.
- Wrappers for OpenAI Gym and Petting Zoo available.
- Because of inheritance an Environment object can also be treated as a single Reward/Done/Broken function.

Hint for developers: only the blue constants, attributes and methods need to be implemented.

#### Set Item Methods:

- `_setup_spaces()` needs to be implemented to enrich the state and action space with specific dimensions.
- The random seed, latency, and mode of the environment can be set explicitly by calling the respective functions.

#### Action Processing Methods:

- When the mode is set to Real, the `process_action` method will forward the Action input to `_export_action` and wait for the feedback received from `_import_state`.
- When the mode is set to Sim, the `process_action` method will forward the Action input to `simulate_reaction` and store the new state using `_set_state`.
- The `process_action` method then continues by computing the reward, done, broken and goal achievement implemented in the respective functions.
- The reset method should reset the environment to initial state.

#### Get Item Methods:

- Mode defines whether the environment is Simulated or Real.
- Cycle Limit defines the limit for training episodes.

Hint for developers: The following methods will return the respective information.

#### Logging Methods:

- Logging can be switched using `switch_logging` method
- The items can be logged manually using the `log` method

#### Plotting Methods:

- The plotting area should be initialized in the `init_plot` method.
- The `update_plot` method should be implemented so that the plotting area is updated.

#### Setup Steps:

There are plenty of methods present, but by following the green bubbles, the environment will be set up and be ready to use.

1. Create your own class and inherit this class.

2. Setup state and action space here.

3. Add the simulation code here or inside an AdaptiveFunction

4. If you want to control real hardware, just implement these two methods

5. The environment is ready to be paired with an agent inside a scenario!

```

class Environment (EnvBase, Mode)
    C_TYPE = 'Environment'
    C_NAME = '????'
    C_CYCLE_LIMIT = 0
    C_LATENCY = timedelta(0,1,0)
    C_REWARD_TYPE = Reward.C_TYPE_OVERALL

    __init__( p_mode=Mode.C_MODE_SIM,
              p_latency=None,
              p_afct_strans=None,
              p_afct_reward=None,
              p_afct_success=None,
              p_afct_broken=None,
              p_logging=Log.C_LOG_ALL )

    STATIC setup_spaces(): MSpace, MSpace
    set_random_seed( p_seed=None )
    set_latency( p_latency:timedelta=None )
    set_mode( p_mode )
    reset( p_seed=None )
    process_action( p_action:Action ): bool
    _set_state( p_state )
    simulate_reaction( p_state:State, p_action:Action ): State
    compute_reward( p_state_old:State,
                   p_state_new:State ): Reward
    compute_success( p_state:State ): bool
    compute_broken( p_state:State ): bool
    clear_buffer()
    _reset( p_seed=None )
    _process_action( p_action:Action ): bool
    _simulate_reaction( p_state:State, p_action:Action ): State
    _compute_broken( p_state:State ): bool
    _compute_success( p_state:State ): bool
    _compute_reward( p_state_old:State,
                   p_state_new:State ): Reward
    _export_action( p_action )
    _import_state()

    get_mode()
    get_latency()
    get_cycle_limit()
    get_action_space()
    get_state_space()
    get_state()
    get_reward_type()
    get_success()
    get_broken()
    get_last_reward()
    get_functions(): AFctSTrans, AFctReward, AFctSuccess,
                  AFctBroken
    switch_logging( p_logging=True )
    log( p_type, *p_args )
    init_plot( p_figure=None )
    update_plot()
  
```

- Environment Creation for Simulation Mode

To create an environment that satisfies MLPro interface is immensely simple and straightforward. Basically a MLPro environment is a class with 5 main functions. Each environment must apply the following mlpro functions:

```
from mlpro.rl.models import *

class MyEnvironment(Environment):
    """
    Custom Environment that satisfies mlpro interface.
    """
    C_NAME          = 'MyEnvironment'
    C_LATENCY       = timedelta(0,1,0)      # Default latency 1s
    C_REWARD_TYPE   = Reward.C_TYPE_OVERALL # Default reward type

    def __init__(self, p_mode=C_MODE_SIM, p_latency:timedelta=None, p_logging=True):
        """
        Parameters:
            p_mode          Mode of environment (simulation/real)
            p_latency       Optional: latency of environment. If not
        provided
                           internal value C_LATENCY will be used by
        default
            p_logging       Boolean switch for logging
        """
        super().__init__(p_latency=p_latency, p_logging=p_logging)
        self._setup_spaces()
        self.set_mode(p_mode)

    def _setup_spaces(self):
        """
        Implement this method to enrich the state and action space with
        specific
        dimensions.
        """
        # Setup state space example
        # self.state_space.add_dim(Dimension(0, 'Pos', 'Position', "", 'm', 'm',
        [-50,50]))
        # self.state_space.add_dim(Dimension(1, 'Vel', 'Velocity', "", 'm/sec', '\
        frac{m}{sec}', [-50,50]))

        # Setup action space example
        # self.action_space.add_dim(Dimension(0, 'Rot', 'Rotation', "", '1/sec', '\
        frac{1}{sec}', [-50,50]))
        ....

    def _simulate_reaction(self, p_action:Action) -> None:
        """
        Simulates a state transition of the environment based on a new
        action.
```

(continues on next page)



(continued from previous page)

```

Please use method set_state() for internal update.

Parameters:
    p_action      Action to be processed
    """
    . . . .

def reset(self) -> None:
    """
    Resets environment to initial state.
    """
    . . . .

def compute_reward(self) -> Reward:
    """
    Computes a reward.

    Returns:
        Reward object
    """
    . . . .

def _evaluate_state(self) -> None:
    """
    Updates the goal achievement value in [0,1] and the flags done and
    ↪broken
    based on the current state.
    """

    # state evaluations example
    # if self.done:
    #     self.goal_achievement = 1.0
    # else:
    #     self.goal_achievement = 0.0
    . . . .

```

One of the benefits for MLPro users is the variety of reward structures, which is useful for Multi-Agent RL and Game Theoretical approach. Three types of reward structures are supported in this framework, such as:

1. **C\_TYPE\_OVERALL** as the default type and is a scalar overall value
2. **C\_TYPE EVERY\_AGENT** is a scalar for every agent
3. **C\_TYPE EVERY\_ACTION** is a scalar for every agent and action.

#### • Environment Creation for Real Hardware Mode

In MLPro, we can choose simulation mode or real hardware mode. For real hardware mode, the creation of an environment is very similar to simulation mode. You do not need to define **\_simulate\_reaction**, but you need to replace it with **\_export\_action** and **\_import\_state** as it is shown in the following:

```
from mlpro.rl.models import *
```

(continues on next page)

(continued from previous page)

```

class MyEnvironment(Environment):
    """
    Custom Environment that satisfies mlpro interface.
    """
    C_NAME          = 'MyEnvironment'
    C_LATENCY       = timedelta(0,1,0)      # Default latency 1s
    C_REWARD_TYPE   = Reward.C_TYPE_OVERALL # Default reward type

    def __init__(self, p_mode=C_MODE_REAL, p_latency:timedelta=None, p_
    logging=True):
        """
        Parameters:
            p_mode          Mode of environment (simulation/real)
            p_latency       Optional: latency of environment. If not
    provided
                            internal value C_LATENCY will be used by
    default
            p_logging       Boolean switch for logging
        """
        super().__init__(p_latency=p_latency, p_logging=p_logging)
        self._setup_spaces()
        self.set_mode(p_mode)

    def _setup_spaces(self):
        """
        Implement this method to enrich the state and action space with
    specific
        dimensions.
        """
        # Setup state space example
        # self.state_space.add_dim(Dimension(0, 'Pos', 'Position', "", 'm', 'm',
    [-50,50]))
        # self.state_space.add_dim(Dimension(1, 'Vel', 'Velocity', "", 'm/sec', '\
    frac{m}{sec}', [-50,50]))

        # Setup action space example
        # self.action_space.add_dim(Dimension(0, 'Rot', 'Rotation', "", '1/sec', '\
    frac{1}{sec}', [-50,50]))
        ....

    def _export_action(self, p_action:Action) -> bool:
        """
        Exports given action to be processed externally (for instance by a
    real hardware).

        Parameters:
            p_action        Action to be exported

        Returns:
            True, if action export was successful. False otherwise.

```

(continues on next page)

(continued from previous page)

```

        """
        . . . .

    def _import_state(self) -> bool:
        """
        Imports state from an external system (for instance a real
        ↪ hardware).
        Please use method set_state() for internal update.

        Returns:
            True, if state import was successful. False otherwise.
        """
        . . . .

    def reset(self) -> None:
        """
        Resets environment to initial state.
        """
        . . . .

    def compute_reward(self) -> Reward:
        """
        Computes a reward.

        Returns:
            Reward object
        """
        . . . .

    def _evaluate_state(self) -> None:
        """
        Updates the goal achievement value in [0,1] and the flags done and
        ↪ broken
        based on the current state.
        """

        # state evaluations example
        # if self.done:
        #     self.goal_achievement = 1.0
        # else:
        #     self.goal_achievement = 0.0
        . . . .

```

- **Environment from Third Party Packages**

Alternatively, if your environment follows Gym or PettingZoo interface, you can apply our relevant useful wrappers for the integration between third party packages and MLPro. For more information, please click [here](#).

- **Environment Checker**

To check whether your developed environment is compatible to MLPro interface, we provide a test script using unittest. At the moment, you can find the source code [here](#). We will prepare a built-in testing module in MLPro, show you how to execute the testing soon and provides an example as well.

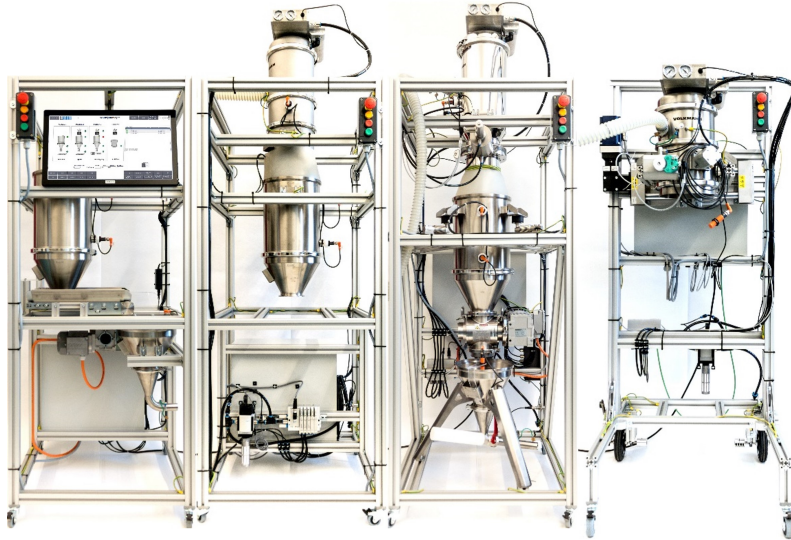
## 6.3.2 Reusing Environment from the Pool

### Bulk Good Laboratory Plant (BGLP)

Ver. 2.3.2 (2023-08-22)

This module provides an RL environment of Bulk Good Laboratory Plant (BGLP).

The BGLP illustrates a smart production system with high flexibility and distributed control to transport bulk raw materials. One of the advantages of this laboratory test belt is the modularity in design, as depicted schematically below:



The BGLP consists of four modules, which are loading, storing, weighing, and filling stations respectively, and has conveying and dosing units as integral parts of the system. The interface between the modules is assembled via a mini hopper placed in the prior module. Then, the next module is fed by a vacuum pump, which operates in a discontinuous manner, before the goods are temporarily stored in a silo of the next module. The filling station has no silo because the main purpose of the station is to occupy the transport containers.

We utilize dissimilar actuators in modules 1-3 to transport the goods from the silo to the mini hopper. Module 1 utilizes a belt conveyor, that operates between 0 and 1800 rpm. Module 2 uses a vibratory conveyor, which can be completely switched on and off. Lastly, Module 3 utilizes a rotary feeder, that operates between 0 and 1450 rpm.

In the RL context, we consider the BGLP as a multi-agent system, where each actuator of the system is pointed as an agent or a player. The states information for each agent is the fill level of the prior reservoir and the fill level of the next reservoir.

---

**Note:** In this simulation, we assume that the actuator in Module D has a constant flow, which automatically matches the production demand in L/s. This parameter can be defined while setting up the BGLP environment. Therefore, 5 actuators are involved in this simulation instead of 6 actuators.

---

The BGLP environment can be imported via:

```
from mlpro.rl.pool.envs.bglp import BGLP
```

**Prerequisites** Please install below package to use the MLPro's BGLP environment

- NumPy

## General Information

Parameter	Value
Agents	5
Native Source	MLPro
Action Space Dimension	[5,]
Action Space Base Set	Real numbers, except Agent 3 uses Integer
Action Space Boundaries	[0,1]
State Space Dimension	[6,]
State Space Base Set	Real numbers
State Space Boundaries	[0,1]
Reward Structure	Individual reward for each agent

**Note:** You can change the configurations of the BGLP simulation, for instance, production demand (L/s), production target for batch operation (L), learning rates for reward calculation, and production scenario (batch or continuous). Batch production scenario refers to a process to satisfy a specific order in a sequence, thus the production target in L must be set. Meanwhile, continuous production scenario refers to a process to control a constant flow within a horizon, thus the production target (L) is not necessary and the target is fulfilled the production demand (L/s). The detailed explanations are available in the API reference section, see here.

## Action Space

In this environment, we consider 5 actuators to be controlled. Thus, there are 5 agents and 5 joint actions because each agent requires an action. Every action is normalized within a range between 0 and 1, except for Agent 3. 0 means the minimum possible action and 1 means the maximum possible action. For Agent 3, the vibratory conveyor has a different character than other actuators, which mostly perform in a continuous manner. The vibratory conveyor can

only be either fully switched-on or switched-off. Therefore the base set of action for Agent 3 is an integer (0/1). 0 means off and 1 means on.

Agent	Actuator	Station	Parameter	Boundaries
1	Conveyor Belt	A	rpm	450 ... 1800
2	Vacuum Pump	B	on-duration (sec)	0 ... 4.575
3	Vibratory Conveyor	B	on/off	0/1
4	Vacuum Pump	C	on-duration (sec)	0 ... 9.5
5	Rotary Feeder	C	rpm	450 ... 1450

### State Space

The state information in the BGLP is the fill levels of the reservoirs. Each agent is always placed in between two reservoirs, e.g. between a silo and a hopper or vice versa. Therefore, each agent has two state information, which is shared with their neighbours. Every state is normalized within a range between 0 and 1. 0 means the minimum fill-level and 1 means the maximum fill-level.

Agent	State No.	Element	Station	Boundaries
1	1	Silo	A	0 ... 17.42 L
	2	Hopper	A	0 ... 9.1 L
2	1	Silo	B	0 ... 17.42 L
	2			
3	1	Hopper	B	0 ... 9.1 L
	2			
4	1	Silo	C	0 ... 17.42 L
	2			
5	1	Hopper	C	0 ... 9.1 L
	2			

### Reward Structure

The reward structure is implemented according to [this paper](#). You can also find the source code of the reward structure, [here](#). The given reward is an individual scalar reward for each agent. To be noted, this reward function is more suitable for a continuous production scenario.

If you would like to implement a customized reward function, you can follow these lines of codes:

```
class MyBGLP(BGLP):

    def calc_reward(self):

        # Each agent has an individual reward
        if self.reward_type == Reward.C_TYPE EVERY_AGENT:
            for actnum in range(len(self.acts)):
                acts = self.acts[actnum]
                self.reward[actnum] = 0
            return self.reward[:]

        # Overall reward
        elif self.reward_type == Reward.C_TYPE OVERALL:
            self.overall_reward = 0
            return self.overall_reward
```

## Cross Reference

- [API Reference](#)

## Citation

If you apply this environment in your research or work, please [cite](#) us and the [original paper](#).

## Multi-Cartpole

The multicartpole environment is an extension over the [cartpole-v1](#) environment native to [OpenAI Gym](#) environments, where a cart is sliding over a flat surface and a pole is attached to the the middle of the cart at one end with frictionless turning joint. With multicartpole environment, we provide you the possibility to simulate multiple cartpole-v1 environments from gym. The goal of this environment is to maintain vertical position of the pole on the cart and stopping it from falling over with the aid of pushing the cart to left or right. The multicartpole environment is visualized in the image below

This multicartpole environment can be imported via:

```
from mlpro.rl.pool.envs.multicartpole import MultiCartPole
```

The multicartpole environment can simulate 'n' number of cartpole-v1 environments simultaneously, where the parameter 'n' can be set while instantiating the environment. The multicartpole environment can be instantiated as an mlpro environment class by including

```
env = MultiCartPole(p_num_envs=3, p_logging=p_logging)
```

## Screenshots

The multicartpole environment consists of 'n' number of internal cartpole-v1 gym environments running simultaneously. The environment starts with random state values and the agent computes actions based on the policy. As there are multiple sub-environments running simultaneously, MLPro offers agent object of type [multi-agent](#), where a number of agents simultaneously simulate corresponding sub-environments. The agent computes an action value of 1 or 0 which refers to a left or right push respectively to the cart. These actions computed by the agents are processed in the corresponding gym sub environment through the MLPro to Gym wrapper functionality of MLPro. The output from the gym sub-environments is the set of new state values and the state flags including success, done, error. The new state of the multicartpole environment is a set of states of all internal sub-environments. The terminal state of multicartpole environment reaches when all the sub-environments are at a terminal state. The sub-environment which are terminal before the rest of the sub-environments the sub-environment is frozen until the rest of the sub-environments are frozen. For better understanding of the multi-cartpole environment and its implementation refer to this example implementation. Running this example implementation of multi-cartpole environment will produce visualisation as in the image below

## Prerequisites

For the multicartpole environment to run properly please install the following python packages:

- [NumPy](#)
- [Matplotlib](#)
- [OpenAI Gym](#)

## General Information

Parameter	Value per sub-environment
Agents	1
Native Source	MLPro
Action Space Dimension	[2,]
Action Space Base Set	Integer number
Action Space Boundaries	[0,1]
State Space Dimension	[4,]
State Space Base Set	Real number
Reward Structure	Overall reward

### Action Space

Since the goal of the environment is to maintain the upright position of the cart, the cart is pushed to right or left for every run of the scenario. The action space for the multicartpole environment consists of push actions +1 and 0, denoting push towards right and left respectively. The size of the action space however is directly proportional to the number of child cartpole-v1 environments running within the multicartpole environment, for example a multicartpole environment for 3 sub environments has an action space of size 3.

Action	Value
Push Left	0
Push Right	1

---

**Note:** The action space for muticartpole environment consists of action spaces for all the sub-environments within the environment. Each of the action space actuates the assigned agent or multi-agent for the subenvironment. To know more about the the multi-agent class functionality native to MLPro refer to the appendix section.

---

### State Space

The state space for the muticartpole environment returns state of every subenvironment within the environment including position of cart, velocity of cart, position of angel and the angular velocity of the pole. The states for a single cartpole environment running inside the multicartpole environment can be understood by the table below.

State	Boundaries
Cart Position	[-2.4,2.4]
Cart Velocity	
Angle of pole	[-0.209,0.209]
Angular Velocity of Pole	

The states of the muticartpole environment also return some flags giving additional information about the environment which includes

- Initial: The flag initial is set to true when an environment has been instantiated or has been reset after a successful or unsuccessful scenario run. The intital flag denotes that there are no adaptations made yet.
- Success: The success flag returns true whem a multicartpole environment has successfully run a scenario for a specified number of cycles. To run an environment successfully, the corresponding states of all the sub environments are within the boundaries as specified in the above table for the number of cycles specified. The scenario ends after the maximum number of cycles specified.



- **Broken:** The broken flag return true when the multicartpole environment is unsuccessful to run for the specified number of cycles. The broken state is set to true when the corresponding states of any sub-environments exceeds the state boundaries as mentioned in the table above.
- **Terminal:** The flag terminal state defines end of an episode or end of a successful scenario of the multicartpole environment. The flag terminal is set to true when the either of the flags success or broken are true. The terminal flag is also set to true if the cycle extends the latency time or at the timeout. Once, the terminal flag is set to true, the environment terminates or resets based on the type of run and number of cycles.

More information about these state parameters related to the multi-cartpole environment can be found in the module descriptions.

### Reward Structure

For multicartpole environment, an overall reward is awarded to the multi-agent. In a single sub-environment of cartpole-v1 a reward value of 1 is returned for every successful cycle run, keeping the states within boundaries. Subsequently, the reward awarded by the multi-cartpole environment is the weighted average of the rewards returned by every internal cartpole-v1 environment.

### Cross Reference

- [Howto RL-AGENT-003: Run Multi-Agent with Own Policy](#)
- [Howto RL-AGENT-004: Train Multi-Agent with Own Policy](#)
- [API Reference](#)

### Citation

If you apply this environment in your research or work, please [cite](#) us.

## Grid World

Ver. 2.0.4 (2023-04-12)

This module provides an environment of customizable Gridworld.

Grid World is a very simple environment and suits to someone who just starts to understand Reinforcement Learning or Markov Decision Process.

In this Grid World environment, by default, the agent will be placed in a 2 dimensional grid world with the size of 8x8, tasked to reach the goal through position increment actions. The user can customize the dimension of the grid and decide the maximum number of steps. The agent is represented by number 1 and the goal is represented by number 2, where number 3 means that the agent is reaching the goal. In the latest version of Grid World, we provided the possibilities to set your own or random initial and/or goal positions. Moreover, there are two possible types of actions, such as continuous actions which can reached the goal in one-shot and discrete actions (only for 2-D grid world). The discrete actions consists of 'up', 'right', 'down', and 'left' (or 'north', 'east', 'south', and 'west') respectively. Here is the example of the grid world environment, by default and with random initial and goal states:

```
[[0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 2, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0]]
```

At the moment, we have not incorporated any obstacles or walls, which will be added in the near future. The current implementation shows that if an action lead to a state outside the boundaries, then the state is back to the previous state.

This Grid World environment can be imported via:

```
from mlpro.rl.pool.envs.gridworld import GridWorld
```

### Prerequisites

- NumPy

### General Information

Parameter	Value
Number of agent	1
Native Source	MLPro
Action Space Dimension	Depends on the grid size, e.g. (8, 8), (8, 8, 8), etc.
Action Space Base Set	(Type 1) Real number (Type 2) Integer number
Action Space Boundaries	(Type 1) Depends on grid_size (Type 2) 0 to 3
State Space Dimension	Depends on the grid size
State Space Base Set	Integer number
State Space Boundaries	0 to 3
Reward Structure	Overall reward

### Action Space

There are two types of actions that can be selected in the beginning of the training, such as continuous actions ('C\_ACTION\_TYPE\_CONT') and discrete actions ('C\_ACTION\_TYPE\_DISC\_2D'). At the moment, the discrete action is limited to 2-dimensional grid world.

For continuous action, the action directly affects the location of the agent. The action is interpreted as increments towards the current location value. The dimension depends on the grid\_size parameter. By default, there is a possibility to reach the target in one shot.

For discrete action, there are four possible actions that represented by number 0 to 3, as follows: Number '0' means 'up' or 'north'. Number '1' means 'right' or 'east'. Number '2' means 'down' or 'south'. Number '3' means 'left' or 'west'.

### State Space

The state space is initialized from the grid\_size parameter, which can be set up to however many dimension as needed. For example, the agent can be placed in a two dimensional world with a n x m size, three dimensional world with a n x m x p, or even more, for instance by setting grid\_size = (n,m) or grid\_size = (n,m,p).

Additionally, the initial and goal position can be randomized or predefined.

### Reward Structure

The default reward function is really simple and straight forward, where the reward is 1, if the agent reaches the goal. The reward is 1 minus the euclidean distance between goal states and current states, if the agent has not reached the goal yet.

```
reward = Reward(self.C_REWARD_TYPE)
rew = 1
euclidean_distance = np.linalg.norm(self.goal_pos-self.agent_pos)
if euclidean_distance !=0:
    rew = 1/euclidean_distance
if self.num_step >= self.max_step:
```

(continues on next page)

(continued from previous page)

```
rew -= self.max_step
reward.set_overall_reward(rew.item())
```

### Cross Reference

- [Howto RL-MB-002: MBRL with MPC on Grid World Environment](#)
- [API Reference](#)

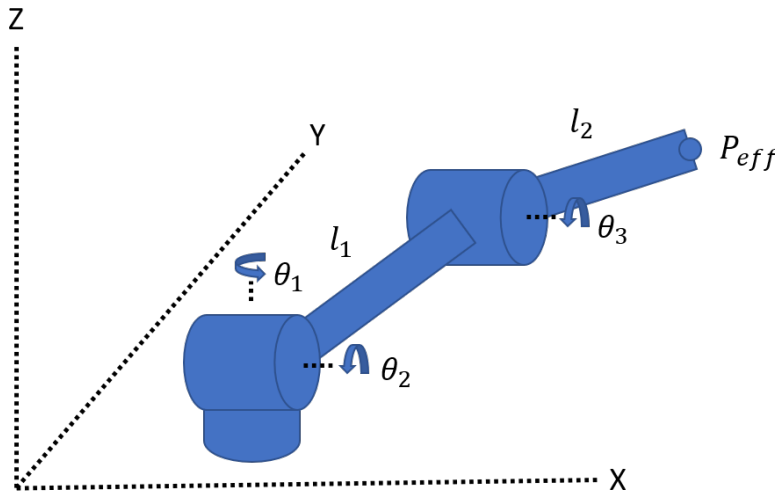
### Citation

If you apply this environment in your research or work, please [cite](#) us.

## Robot Manipulator on Homogeneous Matrix

Ver. 1.1.9 (2023-08-21)

This module provides an environment of a robot manipulator based on Homogeneous Matrix



This environment represents the robot manipulator in term of mathematical equations. The mathematical equations are based on rigid body transformation. In this case, the Homogeneous Transformation Matrix (HTM) is used for the structure. HTM is a matrix that contains both the translation rotation of a point with respect to some plane.

$$H = \begin{bmatrix} \text{Rot} & \text{Trans} \\ \mathbf{0} & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{I} & \text{Trans} \\ \mathbf{0} & 1 \end{bmatrix}}_{\text{translation}} \underbrace{\begin{bmatrix} \text{Rot} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}}_{\text{rotation}}$$

This robotinhtm environment can be imported via:

```
from mlpro.rl.pool.envs.robotinhtm import RobotHTM
```

### Prerequisites

- [NumPy](#)
- [PyTorch](#)

## General Information

Parameter	Value
Agents	1
Native Source	MLPro
Action Space Dimension	[4,]
Action Space Base Set	Real number
Action Space Boundaries	[-pi,pi]
State Space Dimension	[6,]
State Space Base Set	Real number
State Space Boundaries	[-inf,inf]
Reward Structure	Overall reward

### Action Space

By default, there are 4 action in this environment. The action space represents the angular velocity of each joint of the robot manipulator.

### State Space

The state space consists of end-effector positions (x,y,z) of the robot manipulator and target positions (x,y,z).

### Reward Structure

By default, the reward structures are shown in the following equation:

$$reward = -1 * \frac{distError}{initDist} - stepReward$$

### Cross Reference

- [Howto RL-ENV-002: SB3 Policy on RobotHTM Environment](#)
- [Howto RL-MB-001: MBRL on RobotHTM Environment](#)
- [API Reference](#)

### Citation

If you apply this environment in your research or work, please [cite](#) us and the [original paper](#).

## Double Pendulum

Ver. 3.0.0 (2023-05-30)

The Double Pendulum environment is an implementation of a classic control problem of Double Pendulum system. The dynamics of the system are based on the [Double Pendulum](#) implementation by [Matplotlib](#). The double pendulum is a system of two poles, with the inner pole connected to a fixed point at one end and to outer pole at other end. The native implementation of Double Pendulum consists of an input motor providing the torque in either directions to actuate the system.

---

### Note:

**MLPro provides two implementations of Double Pendulum environment named DoublePendulumS4 and DoublePendulumS7.**

- The DoublePendulumS4 environment is a basic implementation with four dimensional state space including angles and angular velocities of both the poles.

- The static 7 dimensional implementation of Double Pendulum environment in MLPro is a seven dimensional state space with derived angular acceleration values and input torque. MLPro also provides a default reward strategy based on normalized state space and Euclidean Distances of the states.

The double pendulum environment can be imported via:

```
from mlpro.rl.pool.envs.doublependulum import *
```

The environment can be initialised with specifying the initial angles of both poles, masses of both poles, lengths of poles, maximum torque value and scenario related parameters including step size and actuation step size. The initial positions of the poles refer to the position of the poles at the beginning of each RL episode, which can be set to 'up', 'down', 'random'. The default values for length and mass of each pole in the double pendulum are set to 1 and 1 respectively. The environment behaviour can be understood by running How To 20 in MLPro's sample implementation examples.

#### Note:

- The visualisation of the environment can be turned off by setting the visualize parameter in training/scenario initialisation to false

## Screenshots

**Prerequisites** Please install below packages to use the MLPro's double pendulum environment

- NumPy
- Matplotlib
- SciPy

## General Information

Parameter	Value
Agents	1
Native Source	MLPro
Action Space Dimension	1
Action Space Base Set	Real number
State Space Dimension	4 (for DoublePendulumS4), 7 (for DoublePendulumS7)
State Space Base Set	Real number
Reward Structure	Overall reward

**Action Space** The goal of the environment is to maintain the vertical position of both the poles. The inner pole is actuated by a motor, and thus the action space of Double Pendulum environment is a continuous variable ranging between the negative maximum torque and positive maximum torque, where positive torque refers to clockwise torque and vice versa. The max torque can be passed as a *parameter* in the initialisation of environment.

Parameter	Range
Torque	[-max_torque, max_torque]

## State Space

The state space for the double pendulum environment returns state of poles in the system including angles of both poles, velocity of poles, angular acceleration of the poles. The states for double pendulum environment can be understood by the table below.

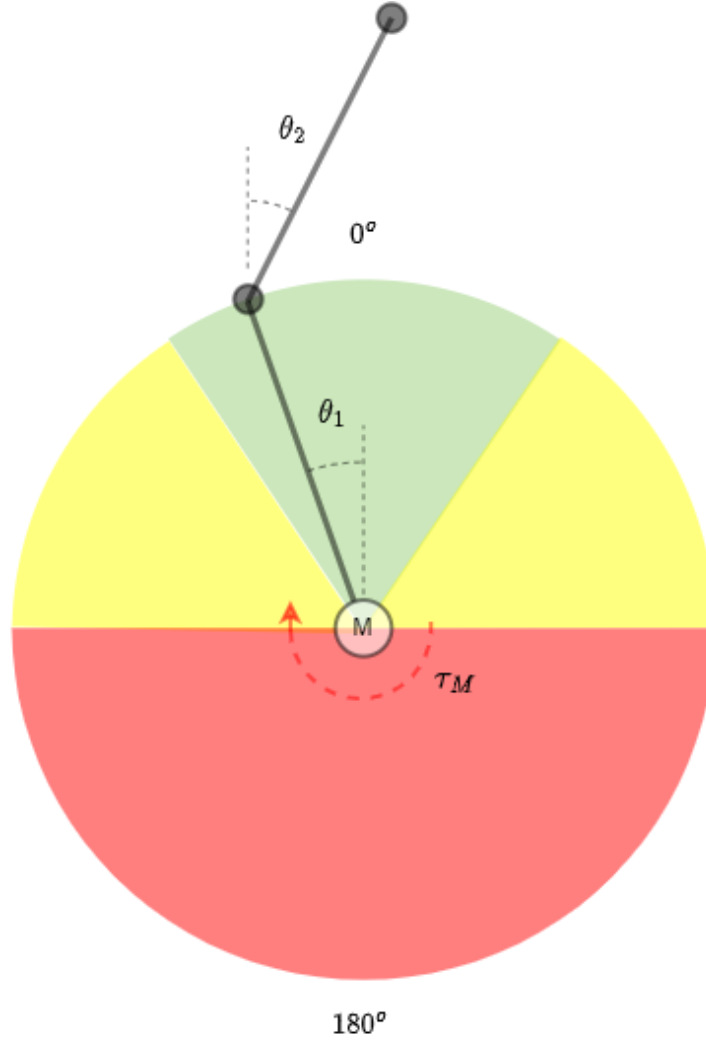
State	Description	Range	Unit	DoublePendulumS4	DoublePendulumS7
Theta 1	Angle of the inner pole	[-180, 180]	degrees	X	X
Omega 1	Angular velocity of inner pole	[-800, 800]	degrees per second	X	X
Alpha 1	Angular Acceleration of outer pole	[-6800, 6800]	degrees per second squared	-	X
Theta 2	Angle of the outer pole	[-180, 180]	degrees	X	X
Omega 2	Angular velocity of outer pole	[-950, 950]	degrees per second	X	X
Alpha 2	Angular acceleration of outer pole	[-9700, 9700]	degrees per second squared	-	X
Torque	Input torque to the inner pole	[-max torque, max torque]	Newton times meter	-	X

**Note:** The boundaries for the velocity and acceleration are highly influenced by the initial position of the arms and the current torque being actuated on the inner pole. These parameters are further dependent on the specific application, scenario or purpose of research.

Current implementation of DP environment in MLPro returns success when the current state of the environment is within a distance lesser than threshold distance from the goal state.

### Reward Structure

The goal of the environment is to reach a complete vertical position for both the inner and outer pole, i.e. the goal state is given as vector  $S_g = (0, 0, 0, 0, 0, 0)$ . The environment delivers a continuous reward to the agent based on the new and old states of the environment. The environment is divided into three zones based on the position of the inner and outer pole.



As shown in the figure above, the three zones and the reward strategies corresponding to the zone are:

1. **Red Zone** : The swing up zone for angle of inner pole less than  $-90^\circ$  or more than  $+90^\circ$ . The reward signal in this zone maximizes the motion of the inner pole of the double pendulum.

$$r_a(t) = (|\theta_{1n(t-1)} - \theta_{1n(t)}|) + (|\theta'_{1n(t)} + \theta_{1n(t-1)}^n| - |\theta'_{1n(t-1)} + \theta_{1n(t-1)}^n|)$$

where,

$r_a(t)$  is reward at time step t,

$\theta_{1n}$  is normalized angle of inner pole

$\theta_{2n}$  is normalized angle of outer pole

2. **Yellow Zone** : Outer pole swing up zone for angle of inner pole more than  $-90^\circ$  or less than  $+90^\circ$ . The reward is based on the euclidean distance between new and old states, with 75% weight to the states of outer pole and 25% to that of inner pole.

$$r_b(t) = |s_{gb} - s_{b(t-1)}| - |s_{gb} - s_b(t)|$$

where,

$s_b$  is the state space in yellow zone as  $(\theta_{1n}, \theta_{2n}, \theta'_{2n}, \theta''_{2n})$

$s_{gb}$  is the goal state in Yellow zone, i.e.  $(0, 0, 0, 0)$

3. **Green Zone** : Balancing zone for angle of either or both inner or outer pole more than  $-36^\circ$  or less than  $+36^\circ$ . The reward in this zone is proportional to the environments progress towards the goal state.

$$r_a(t) = |s_{gn} - s_{n(t-1)}| - |s_{gn} - s_{n(t)}|$$

where,

$s_{gn}$  is the normalized goal state of the environment

$s_n$  is the normalized state

### Cross Reference

- [Howto RL-ENV-005: SB3 Policy on Double Pendulum Environment](#)
- [API Reference](#)

### Citation

If you apply this environment in your research or work, please [cite](#) us.

Alternatively, you can also reuse available environments from 3rd-party packages via wrapper classes (currently available: OpenAI Gym or PettingZoo).

For reusing the 3rd packages, we develop a wrapper technology to transform the environment from the 3rd-party package to the MLPro-compatible environment. Additionally, we also provide the wrapper for the other way around, which is from MLPro Environment to the 3rd-party package. At the moment, there are two ready-to-use wrapper classes. The first wrapper class is intended for OpenAI Gym and the second wrapper is intended for PettingZoo. The guide to using the wrapper classes is step-by-step explained in our how-to files, as follows:

- (1) OpenAI Gym to MLPro,
- (2) MLPro to OpenAI Gym,
- (3) PettingZoo to MLPro, and
- (4) MLPro to PettingZoo.

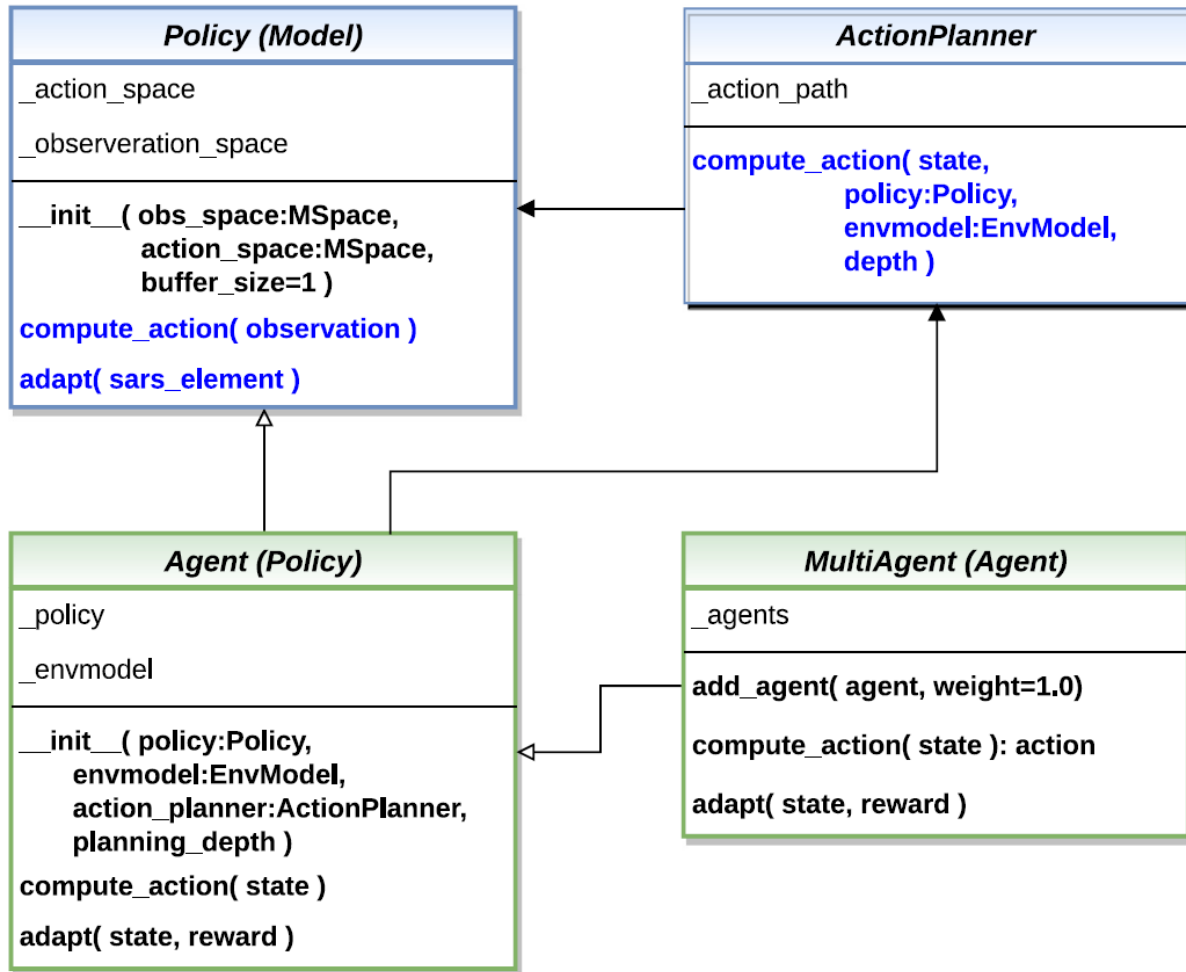
## 6.4 Agents

In RL, an agent is an autonomous entity that interacts with an environment, receiving rewards for performing certain actions and updates its behavior based on that feedback. The agent's goal is to learn a policy that maximizes its cumulative reward over time.

From a scientific perspective, the agent is typically modeled as a decision-making system that maps states of the environment to actions through a policy. The policy can be deterministic or probabilistic and can be learned through various RL algorithms such as Q-Learning, SARSA, or Policy Gradient methods. The agent's performance is evaluated using metrics such as reward, cumulative reward, and value functions. Overall, an agent in RL provides an algorithm that makes decisions and learns from experience to optimize its performance in a given task.

MLPro-RL supplies a special agent model landscape, which covers different RL scenarios including a simple single-agent RL, a multi-agent RL, and model-based agents with an optional action planner. For the multi-agent RL, the structure is constructed by assigning multiple single agents in a group. The main component of each single-agent (either single-agent or multi-agent RL) is the policy. The basic class of the policy is inherited from the ML Model of basic MLPro functionality and extended by the RL-related function of the action calculation. The users can inherit the basic class of the policy to implement their *own custom algorithms* or simply use algorithms from third-party packages via wrapper classes. The other possibility would be *importing algorithms from the pool object*. For an overview, the simplified class diagram of agents in MLPro is described below.



Fig. 3: This figure is taken from [MLPro 1.0 paper](#).

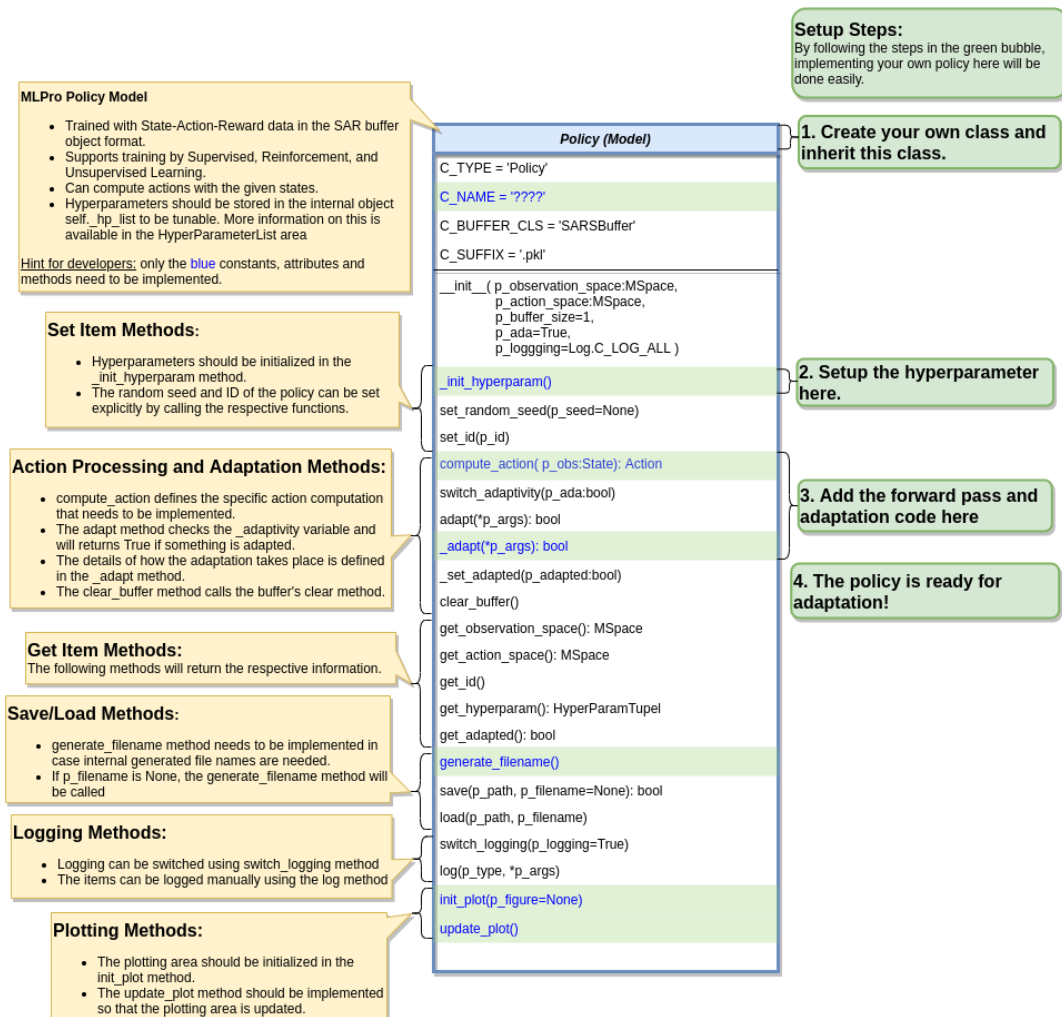
Moreover, an environment model (known as *EnvModel class*) can be supplemented to a single agent, i.e. for model-based RL cases. This class can be used for model-based learning, which learns the behaviour or dynamics of the environment. Another possible extension of the model-based agent is an action planner. Action planner uses the environment model (or EnvModel) to plan the next action by predicting the output on a certain horizon. An example of action planner algorithms is *Model Predictive Control (MPC)*, which is also provided in MLPro.

Additionally, you can find more comprehensive explanations of agents in MLPro-RL including a sample application on controlling a UR5 Robot in this paper: [MLPro 1.0 - Standardized Reinforcement Learning and Game Theory in Python](#).

Here are some subsections of the agent model landscape of MLPro-RL, which might be interesting for the users:

### 6.4.1 Custom Policies

#### • Policy Creation



Creating a custom RL policy that satisfies the MLPro interface is straightforward. First of all, the users need to inherit a base Policy class. Then, the users can develop their custom policies by fulfilling at least 2 main functions, namely **compute\_action** and **\_adapt**, as shown in the following code. The compute action method (**compute\_action**) is a function to calculate an action in the current state. Meanwhile, the adapt method (**\_adapt**) is a function to optimize the policy according to past experience.

```

from mlpro.rl.models import *

class MyPolicy (Policy):
    """
    Creates a policy that satisfies mlpro interface.
    """

    C_NAME      = 'MyPolicy'

    def compute_action(self, p_state: State) -> Action:
        """
        Specific action computation method to be redefined.

        Parameters:
            p_state      State of environment

        Returns:
            Action object
        """
        ....

    def _adapt(self, *p_args) -> bool:
        """
        Adapts the policy based on State-Action-Reward (SAR) data that will
        ↪ be expected as a SAR
        ↪ buffer object. Please call super-method at the beginning of your
        ↪ own implementation and
        ↪ adapt only if it returns True.

        Parameters:
            p_arg[0]      SAR Buffer object
        """

        if not super().adapt(*p_args): return False

        ....
        return True

```

Hyperparameters of the policy should be stored in the internal object **self.\_hp\_list**, so that they can be tuned from outside. The hyperparameter initialization method (**\_init\_hyperparam**) can be used in this case. To set up a hyperparameter space, please refer to our [how-to file](#).

- **Policy from Third Party Packages**

Alternatively, the user can also apply algorithms from Stable Baselines 3 by using the developed relevant wrapper for the integration between third-party packages and MLPro. For more information, please click [here](#).

- **Algorithm Checker**

A test script using a unit test to check the developed policies will be available soon!

## 6.4.2 Policy Pool

### Random Action Generator

Ver. 1.0.5 (2023-09-21)

This module provides random generator for multi purposes, e.g. testing environment, etc..

#### General Information

A random generator for a specific action space with defined boundaries is an algorithm that generates random values within a specific range or set of constraints. This type of random generator is often used for testing environments, generate sample data for model-based learning, and many more. The random generator can be attach to an RL agent and will detect the action space of the agent as well as the boundaries. The action space refers to the set of possible actions of the agent, and the boundaries define the range of values within which the actions must lie. Then, an action is randomly computed by the generator using uniform distribution.

For example, if the action space of an agent is defined as a joint velocity of a robot, the boundaries may be defined as the minimum and maximum velocity of the joint. The random generator would then generate random values within these boundaries that represent the possible velocity of the robot. The specific method used to generate random values within the action space boundaries depends on the type of data being generated and the desired properties of the generated data.

This Random Action Generator policy can be imported via:

```
from mlpro.rl.pool.policies.randomgenerator import RandomGenerator
```

#### Cross Reference

- [API Reference](#)

#### Citation

If you apply this policy in your research or work, please [cite](#) us.

### Model Predictive Control (MPC)

Ver. 1.1.1 (2023-02-04)

This module provides a default implementation of model predictive control (MPC).

#### Prerequisites

- [NumPy](#)

#### General Information

We introduce an MPC method as action planner in the model-based RL territory. Monte Carlo MPC is a control algorithm that uses a Monte Carlo simulation-based approach to generate control actions for a dynamic system. It is a type of MPC, which is a well-established control algorithm that predicts the future behavior of a system based on a mathematical model and uses this information to generate optimal control actions.

In this Monte Carlo MPC, instead of using a single action prediction of the future system behavior, a large number of simulations are run, each with different random actions variations in the model parameters. Based on these trials, Monte Carlo MPC generates control actions that minimize a defined cost function, taking into account the control objectives. The control actions are updated at each time step based on new measurements of the system state.

Monte Carlo MPC is particularly useful in situations where the system is uncertain, unpredictable, or subject to significant external disturbances, as it allows for a probabilistic treatment of these uncertainties. It has found applications in a variety of fields, including autonomous systems, robotics, and process control.

This MPC policy can be imported via:

```
from mlpro.rl.pool.actionplanner.mpc import MPC
```

Multiprocessing has also been incorporated into MPC, which allows parallel computations. Depending on the number of planning horizon, but we believe that this reduces the training time massively.

#### Cross Reference

- [Howto RL-MB-002: MBRL with MPC on Grid World Environment](#)
- [API Reference](#)

#### Citation

If you apply this policy in your research or work, please [cite](#) us and the [original paper](#).

### 6.4.3 Model-Based Agents

Model-Based Agents have a dissimilar learning target as Model-Free Agents, whereas learning the environment model is not required in the model-free RL. An environment model can be incorporated into a single agent, see EnvModel for an overview. Then, this model learns the behaviour and dynamics of the environment. After learning the environment, the model is optimized to be able to accurately predict the output states, rewards, or status of the environment with respect to the calculated actions. As a result, if the predictions of the subsequent state and reward diverge too far from the actual values of the environment, the environment model itself is incorporated into the agent's adaptation process and is always retrained. An adaptation in the environment model necessitates an adaptation in the policy. The foundation for this is an internal episodic training of the policy in interaction with the environment model.

After having a model that can accurately predict the behaviour of the environment, the single agent is optionally extended as an action planner. The action planner can be used by the environment to plan the next actions, e.g. using Model Predictive Control. In MLPro-RL, we have also provided a base class for ActionPlanner, where only the action planner method (**\_plan\_action**) and an optional custom setup method (**\_setup**) are needed to be adjusted, as shown below:

```
from mlpro.rl.models import *

class MyActionPlanner (ActionPlanner):
    """
    Creates an action planner that satisfies mlpro interface.
    """

    C_NAME      = 'MyActionPlanner'

    def _setup(self):
        """
        Optional custom setup method.
        """

        pass

    def _plan_action(self, p_obs: State) -> SARSBuffer:
        """
        Custom planning algorithm to fill the internal action path (self._action_path).
        ↳ Search width
        and depth are restricted by the attributes self._width_limit and self._

```

(continues on next page)

(continued from previous page)

```

↪prediction_horizon.
    Parameters
    -----
    p_obs : State
        Observation data.
    Returns
    -----
    action_path : SARSBuffer
        Sequence of SARSElement objects with included actions that lead to the best.
↪possible reward.
    """

    raise NotImplementedError

```

### Environment Model (EnvModel)

To set up environment model, the adaptive function needs to be created first. In this case, our adaptive function will predict the next state of the environment based on provided action. After that, we need to create another class that is inherited from the actual environment module and **EnvModel**, in this case RobotHTML. For now, we only use the state transition model. The reward, success and broken model are taken from the original environment module.

```

from mlpro.rl.model_env import EnvModel
from mlpro.rl.pool.envs.robothtml import RobotHTML

class OurEnvModel(RobotHTML, EnvModel):
    C_NAME = "Our Env Model"

    # Put necessary input argument in initialization
    def __init__(
        self,
        p_num_joints=4,
        p_target_mode="Random",
        p_ada=True,
        p_logging=False,
    ):

        # Initialize the actual environment to get all environment functionalities, such.
↪as
        # _simulate_reaction, _reset, _compute_reward, _compute_broken and _compute_
↪success
        RobotHTML.__init__(self, p_num_joints=p_num_joints, p_target_mode=p_target_mode)

        # Setup Adaptive Function
        afct_strans = AFctSTrans(
            OurStatePredictor,
            p_state_space=self._state_space,
            p_action_space=self._action_space,
            p_threshold=1.8,
            p_buffer_size=20000,
            p_ada=p_ada,
            p_logging=p_logging,
        )

```

(continues on next page)

(continued from previous page)

```

    # In this case set only p_afct_strans, which tells the module to use
    # _simulate_reaction from the adaptive function instead of from the actual_
↪environment
    # Set to None to use function such as compute_reward, compute_broken and compute_
↪success
    # from the actual environment
    EnvModel.__init__(
        self,
        p_observation_space=self._state_space,
        p_action_space=self._action_space,
        p_latency=timedelta(seconds=self.dt),
        p_afct_strans=afct_strans,
        p_afct_reward=None,
        p_afct_success=None,
        p_afct_broken=None,
        p_ada=p_ada,
        p_logging=p_logging,
    )

    self.reset()

```

#### Cross Reference

- [Howto RL-MB-001: Train and Reload Model Based Agent \(Gym\)](#)
- [Howto RL-MB-002: MBRL with MPC on Grid World Environment](#)
- [Howto RL-MB-003: MBRL on RobotHTM Environment](#)
- [MLPro-SL](#)

## 6.4.4 Multi-Agents

In Reinforcement Learning, multi-agent refers to a scenario where multiple independent agents interact with each other and with their environment in an attempt to accomplish a common goal or optimize their own reward. In this setting, the behavior of each agent not only depends on its own actions and observations, but also on the actions and observations of other agents, leading to a complex, dynamic and interactive decision-making process. The scientific study of multi-agent reinforcement learning involves modeling the interdependence and cooperation/competition among agents, and developing algorithms for agents to learn and adapt to the environment effectively.

MLPro-RL is not only compatible with single-agent RL but also multi-agent RL, where the extent of the agent landscape is completed by the multi-agent model. It is compatible with single-agent but does not have its own policy. Instead, it is utilized to combine and control any quantity of single agents that together control the action calculation. Every single agent in this situation interacts with a separate portion of the surrounding multi-observation agents and action space. Multi-agent interactions take place in appropriate contexts that support the scalar reward per agent reward type. These are native applications that incorporate the MLPro environment template or PettingZoo environments that may be incorporated using the corresponding wrapper class offered by MLPro.

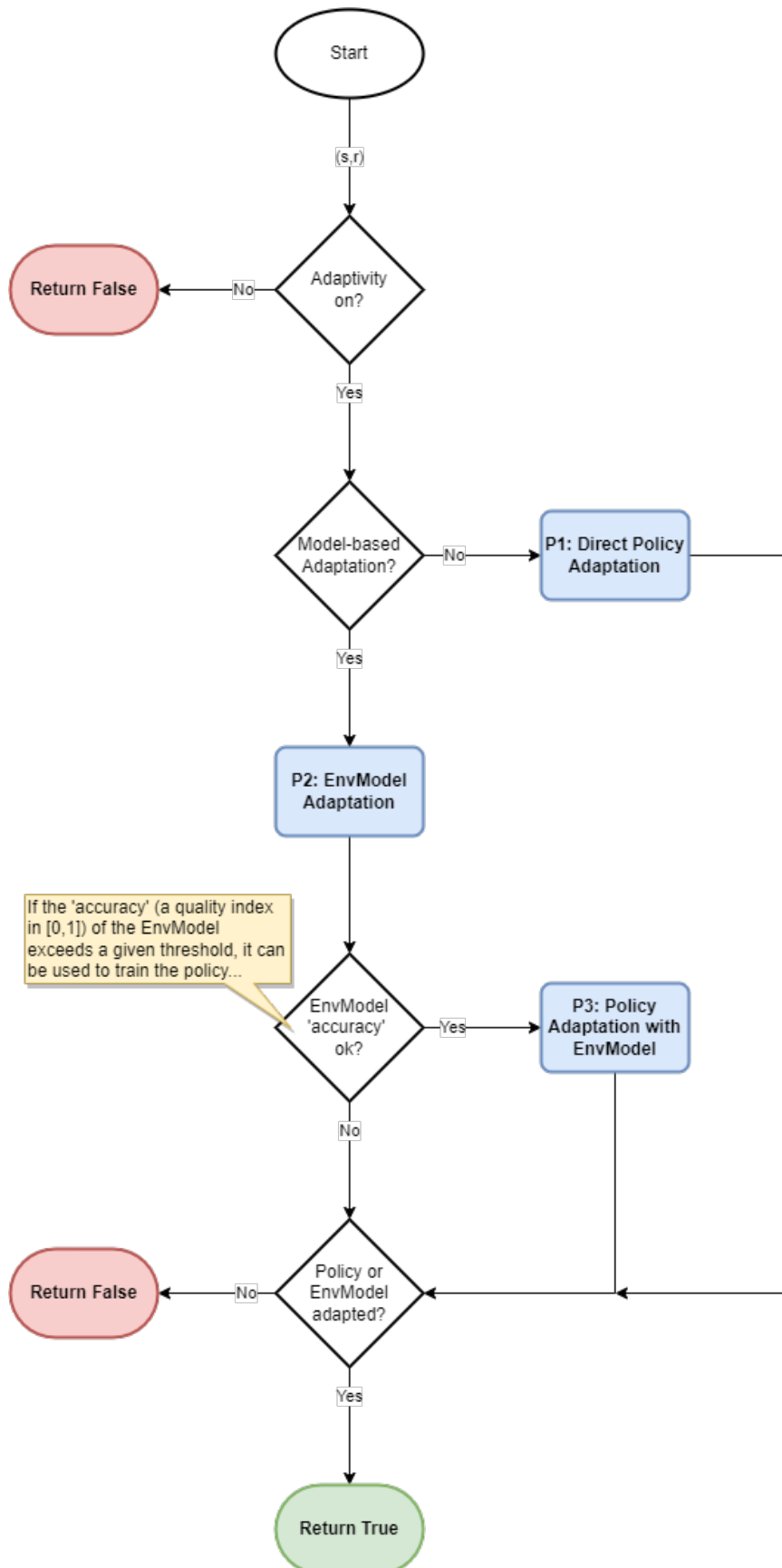
#### Cross Reference

- [Howto RL-AGENT-004: Train Multi-Agent with Own Policy](#)
- [MLPro-RL: Training](#)

The following flowchart describes the adaptation procedure of an agent. In the beginning, the loop checks whether it is model-based RL or model-free RL. If it is a model-free RL, then the loop is jumped to a direct policy adaptation. Then,

the current step ended after the policy adaptation. Meanwhile, in the model-based RL, the EnvModel is first adapted, and then the loop checks whether the accuracy of the EnvModel exceeds a given threshold. This activity is to make sure that the EnvModel is accurate enough for policy adaptation. If the accuracy is higher than the threshold, then the policy adaptation takes place with EnvModel. Otherwise, the current step is ended without any policy adaptations.





## 6.5 Scenarios

A scenario in reinforcement learning refers to a specific problem or task that the agent is trying to learn how to solve. A scenario defines the environment in which the agent operates, including the state space, the action space, the reward function, and the transition dynamics.

In RL, the agent interacts with the environment over a sequence of time steps. At each time step, the agent receives an observation of the current state of the environment and selects an action. The environment then transitions to a new state and returns a reward signal to the agent.

The scenario provides the agent with a set of goals to be achieved, and the reward function quantifies how well the agent is doing in terms of achieving these goals. The reward function can be designed to encourage certain behaviors, such as reaching a specific target state, or penalize certain behaviors, such as taking actions that lead to a state of low reward.

The scenario also defines the state and action spaces, which are the sets of all possible states and actions that the agent can experience and take, respectively. The transition dynamics describe how the environment changes in response to the agent's actions.

Overall, a scenario in RL defines the problem that the agent is trying to solve, and provides the necessary information for the agent to learn a policy that maps states to actions and maximizes the cumulative reward signal.

In MLPro-RL, a class **RLScenario** inherits the functionality from class **Scenario** in the basic function level, where the **RLScenario** class combines RL agents and an environment into an executable unit.

One of the MLPro's features is enabling the user to apply a template class for an RL scenario consisting of an environment and agents. Moreover, the users can create either a single-agent scenario or a multi-agent scenario in a simple manner by inheriting **RLScenario** base class and redefining its **\_setup** function.

### Cross Reference

- [MLPro-RL: Training](#)
- [Howto RL-AGENT-001: Run an Agent with Own Policy](#)
- [Howto RL-AGENT-003: Run Multi-Agent with Own Policy](#)

## 6.6 Training and Tuning

In RL, the agent and the environment interact over a sequence of time steps. At each time step, the agent receives an observation of the current state of the environment and selects an action. The environment then transitions to a new state and returns a reward signal to the agent. This process continues until some terminal state is reached.

The agent uses the observed state-action-reward sequences to update its policy, either through model-based methods that estimate the underlying dynamics of the environment, or model-free methods that directly estimate the value or the policy. The policy is used to select actions in subsequent interactions with the environment, allowing the agent to learn from its mistakes and improve over time.

In MLPro-RL, a class **RLTraining** inherits the functionality from class **Training** in the basic function level, where the **RLTraining** class are used for training and hyperparameter tuning of RL agents. We implement episodic training algorithms and make the corresponding extended training data and results as well as the trained agents available in the file system. In this RL training, we always start with a defined random initial state of the environment and evaluate at each time step whether one of the following three categories is satisfied,

- (1) **Event Success:** This means that the defined target state is reached and the actual episode is ended.
- (2) **Event Broken:** This means that the defined target state is no longer reachable and the actual episode is ended.

- (3) **Event Timeout:** This means that the maximum training cycles for an episode are reached and the actual episode is ended.

If none of the events is satisfied, then the training continues. The goal of the training is to maximize the score of the repetitive evaluations. In this case, a stagnation detection functionality can be incorporated to avoid a long training time without any more improvements. The training can be ended, once the stagnation is detected. For more information, you can read [Section 4.3 of MLPro 1.0 paper](#).

In MLPro-RL, we simplify the process of setting up an RL scenario and training for both single-agent and multi-agent RL, as shown below:

- **Single-Agent Scenario Creation**

```
from mlpro.rl.models import *

class MyScenario(Scenario):

    C_NAME      = 'MyScenario'

    def _setup(self, p_mode, p_ada:bool, p_logging:bool):
        """
        Here's the place to explicitly setup the entire rl scenario.
        ↪Please bind your env to
           self._env and your agent to self._agent.

        Parameters:
            p_mode           Operation mode of environment (see
        ↪Environment.C_MODE_*)
            p_ada            Boolean switch for adaptivity of agent
            p_logging        Boolean switch for logging functionality
        """

        # Setup environment
        self._env = MyEnvironment(...)

        # Setup an agent with selected policy
        self._agent = Agent(
            p_policy=MyPolicy(
                p_state_space=self._env.get_state_space(),
                p_action_space=self._env.get_action_space(),
                ....
            ),
            ....
        )

        # Instantiate scenario
        myscenario = MyScenario(p_scenario=myscenario, ....)

        # Train agent in scenario
        training = Training(...)
        training.run()
```

- **Multi-Agent Scenario Creation**

```
from mlpro.rl.models import *
```

(continues on next page)

(continued from previous page)

```

class MyScenario(Scenario):

    C_NAME      = 'MyScenario'

    def _setup(self, p_mode, p_ada:bool, p_logging:bool):
        """
        Here's the place to explicitly setup the entire rl scenario.
        Please bind your env to
        self._env and your agent to self._agent.

        Parameters:
            p_mode          Operation mode of environment (see
        Environment.C_MODE_*)
            p_ada            Boolean switch for adaptivity of agent
            p_logging        Boolean switch for logging functionality
        """

        # Setup environment
        self._env = MyEnvironment(...)

        # Create an empty multi-agent
        self._agent = MultiAgent(...)

        # Add Single-Agent #1 with own policy (controlling sub-environment
        #1)
        self._agent.add_agent = Agent(
            self._agent = Agent(
                p_policy=MyPolicy(
                    p_state_space=self._env.get_state_space().spawn[...],
                    p_action_space=self._env.get_action_space().spawn[...],
                    ....
                ),
                ....
            ),
            ....
        )

        # Add Single-Agent #2 with own policy (controlling sub-environment
        #2)
        self._agent.add_agent = Agent(...)

        ....

    # Instantiate scenario
    myscenario = MyScenario(p_scenario=myscenario, ....)

    # Train agent in scenario
    training = Training(...)
    training.run()

```

## Cross Reference

- [A sample application video of MLPro-RL on a UR5 robot](#)
- [Howto RL-AGENT-002: Train an Agent with Own Policy](#)
- [Howto RL-AGENT-004: Train Multi-Agent with Own Policy](#)
- [Howto RL-AGENT-011: Train and Reload Single Agent \(Gym\)](#)
- [Howto RL-AGENT-021: Train and Reload Single Agent \(MuJoCo\)](#)
- [Howto RL-ATT-001: Train and Reload Single Agent using Stagnation Detection \(Gym\)](#)
- [Howto RL-ATT-002: Train and Reload Single Agent using Stagnation Detection \(MuJoCo\)](#)
- [\*Howto RL-MB-001: Train and Reload Model Based Agent \(Gym\)\*](#)
- [Howto RL-MB-002: MBRL with MPC on Grid World Environment](#)
- [\*MLPro-BF-ML: Training and Tuning\*](#)

## 6.7 3rd Party Support

MLPro enables the seamless integration of RL environments and policy algorithms of popular open-source projects into own applications. To this regard, MLPro provides additional integration packages that wrap selected 3rd party functionalities and demonstrate their reuse in numerous example programs. In particular, the following projects are supported:

- [MLPro-Int-SB3 - Integration of Stable Baselines 3 into MLPro](#)
- [MLPro-Int-Gymnasium - Integration of Gymnasium into MLPro](#)
- [MLPro-Int-PettingZoo - Integration of PettingZoo into MLPro](#)

Discover the complete list of integrated 3rd party packages and further extensions in the [MLPro Extension Hub](#).



## MLPRO-GT - GAME THEORY

### 7.1 Overview

Game theory (GT) is a branch of mathematics and economics that studies strategic interactions among rational decision-makers. It has applications in various fields, including economics, engineering, biology, political science, and computer science. In essence, GT provides a framework for analyzing and understanding the behavior of individuals or entities when their decisions affect each other.

In the native GT, at its core, GT models decision-making scenarios where the outcome of an individual's choice depends not only on their actions but also on the actions of others. These scenarios, known as “games,” involve players, strategies, and payoffs. Players make decisions based on their understanding of the strategic environment, aiming to maximize their utility or payoff.

MLPro-GT is designed to empower researchers, analysts, and developers with a comprehensive set of tools for game theory analysis. Python's simplicity, readability, and extensive libraries make it an ideal language for implementing and experimenting with GT models. Our package distinguishes itself by offering two powerful sub-frameworks:

(1) **MLPro-GT-Native:** This framework focuses on modeling and analyzing traditional static or native games. Native games capture strategic interactions where players make decisions simultaneously. Our Python package simplifies the representation and analysis of native games, providing a user-friendly interface to explore solution concepts such as Nash equilibrium.

(2) **MLPro-GT-DG:** DG stands for dynamic games. For scenarios involving sequential decision-making or repeated interactions, our dynamic games framework comes into play. Dynamic GT extends the analysis to capture evolving strategies over time, allowing users to model and simulate complex, dynamic strategic environments.

MLPro-GT-DG is developed as a sub-framework for the cooperative GT approach to solving multi-agent problems by inheriting a handful of main functionalities of MLPro-RL, such as the environment model, the agent model, and the environment-agent interaction model. This sub-framework focuses on the cooperative GT approach to Markov games. A Markov game contains a group of independent players that make decisions simultaneously, see the figure below for an overview.

If you are interested to utilize MLPro-GT-Native and MLPro-GT-DG, you can easily access the GT modules, as follows:

```
from mlpro.gt.native import *  
from mlpro.gt.dynamicgames import *
```

Additionally, you can find more comprehensive explanations of MLPro-GT-DG including a sample application and difference with a native RL approach in this paper: [MLPro 1.0 - Standardized Reinforcement Learning and Game Theory in Python](#).

#### Learn more

- *[Getting started with MLPro-GT](#)*

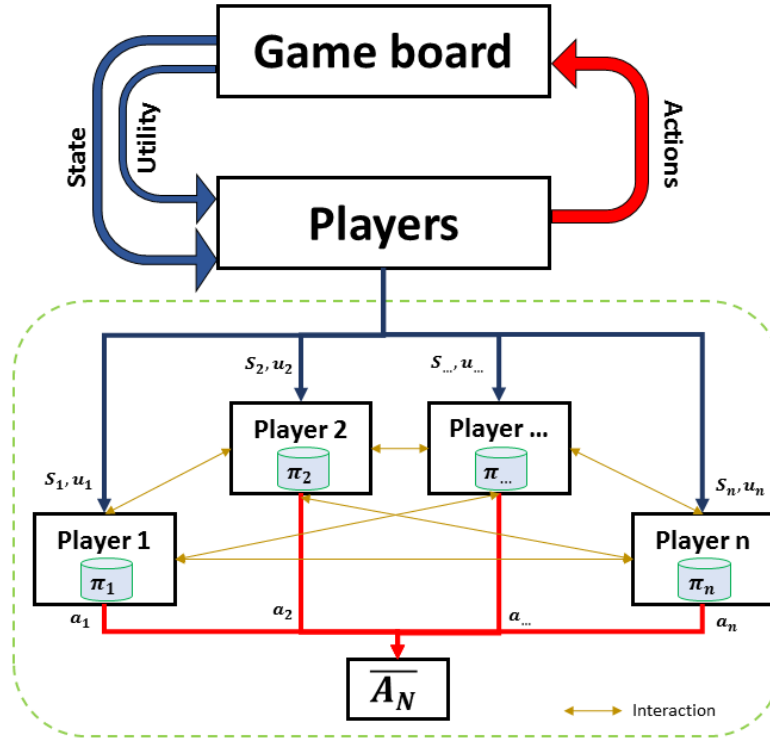


Fig. 1: This figure is taken from MLPro 1.0 paper.

### Cross Reference

- [Related Howtos](#)
- [API Reference: MLPro-GT](#)
- [API Reference: MLPro-GT Pool of Objects](#)
- [MLPro 1.0 Paper](#)

## 7.2 Getting Started

### 7.2.1 MLPro-GT-Native - Native Games

Here is a concise series to introduce all users to the MLPro-GT-Native in a practical way, whether you are a first-timer or an experienced MLPro user.

If you are a first-timer, then you can begin with **Section (1) What is MLPro?**.

If you have understood MLPro but not the game theoretical approach, then you can jump to **Section (2) What is Game Theory?**.

If you have experience in both MLPro and game theory, then you can directly start with **Section (3) What is MLPro-GT?**.

After following the below step-by-step guideline, we expect the user understands the MLPro-GT in practice and starts using MLPro-GT-Native.



### 1. What is MLPro?

If you are a first-time user of MLPro, you might wonder what is MLPro. Therefore, we recommend initially starting with understanding MLPro by checking out the following steps:

- (a) [\*MLPro: An Introduction\*](#)
- (b) [\*introduction video of MLPro\*](#)
- (c) [\*installing and getting started with MLPro\*](#)
- (d) [\*MLPro paper in Software Impact journal\*](#)

### 2. What is Game Theory?

Game theory is a branch of mathematics and social science that examines strategic interactions among rational decision-makers. It involves modeling scenarios where individuals or entities, referred to as players, make decisions that influence one another. Players choose from a set of strategies, and the resulting outcomes, or payoffs, are determined by the collective decisions. The concept of Nash equilibrium, where no player has an incentive to unilaterally deviate from their chosen strategy, is central to game theory. This framework is widely applied across disciplines such as economics, engineering, political science, and biology, offering insights into strategic decision-making, predicting outcomes, and formulating optimal strategies in interactive situations.

### 3. What is MLPro-GT?

We expect that you have a basic knowledge of MLPro and game theory. Therefore, you need to understand the overview of MLPro-GT by following the steps below:

- (a) [\*MLPro-GT introduction page\*](#)
- (b) [\*Section 5 of MLPro 1.0 paper\*](#)

### 4. Understanding Players, Strategies, Payoffs, and Games in MLPro-GT-Native

We provide the definition of each main component of game theory in our documentation page, including howto setup each component. For better understanding, you can direct to the following pages:

- (a) [\*MLPro-GT-Native: Player, Coalition, Competition\*](#)
- (b) [\*MLPro-GT-Native: Payoff\*](#)
- (c) [\*MLPro-GT-Native: Solvers\*](#)
- (d) [\*MLPro-GT-Native: Games\*](#)

### 5. Access HowTo Files related to MLPro-GT-Native

For better understanding of the applications, you can direct to our sample applications in the following page:

- [\*HowTo MLPro-GT-Native\*](#)

## 7.2.2 MLPro-GT-DG - Dynamic Games

Here is a concise series to introduce all users to the MLPro-GT-DG in a practical way, whether you are a first-timer or an experienced MLPro user.

If you are a first-timer, then you can begin with **Section (1) What is MLPro?**.

If you have understood MLPro but not the game theoretical approach in the engineering field, then you can jump to **Section (2) What is Game Theory?**.

If you have experience in both MLPro and game theory, then you can directly start with **Section (3) What is MLPro-GT?**.

After following the below step-by-step guideline, we expect the user understands the MLPro-GT in practice and starts using MLPro-GT-DG.

### 1. What is MLPro?

If you are a first-time user of MLPro, you might wonder what is MLPro. Therefore, we recommend initially starting with understanding MLPro by checking out the following steps:

- (a) *MLPro: An Introduction*
- (b) introduction video of MLPro
- (c) *installing and getting started with MLPro*
- (d) MLPro paper in Software Impact journal

### 2. What is Game Theory?

If you have not dealt with game theory for engineering applications, we recommend starting to understand at least the basic concept of game theory. There are plenty of references, articles, papers, books, or videos on the internet that explains the game theory. But, for deep understanding, we recommend you to read the book from Dario Bauso, which is *Game Theory with Engineering Applications*.

### 3. What is MLPro-GT?

We expect that you have a basic knowledge of MLPro and game theory. Therefore, you need to understand the overview of MLPro-GT by following the steps below:

- (a) *MLPro-GT introduction page*
- (b) Section 5 of MLPro 1.0 paper

### 4. Understanding Game Board and Player in MLPro-GT-DG

First of all, it is important to understand the structure of a game board in MLPro-GT, which can be found on *this page*.

In reinforcement learning, we have two types of agents, such as a single-agent RL or a multi-agent RL. Both of the types are covered by MLPro-RL. Meanwhile, in MLPro-GT, we focus on a multi-player GT because there are no significant advantages of using game theory for single-player. To understand a player in MLPro-GT, you can visit *this page*.

Then, you can start following some of our howto files and a sample application that shows how to run and train multi-player with their own policy, as follows:

- (a) *Howto GT-001: Run Multi-Player with Own Policy*
- (b) *Howto GT-002: Train Multi-Player*
- (c) Section 6.2 of MLPro 1.0 paper

### 5. Additional Guidance

After following the previous steps, we hope that you could practice MLPro-GT and start using this subpackage for your GT-related activities. For more advanced features, we highly recommend you to check out the following howto files:

- (a) Howto RL-HT-001: Hyperopt
- (b) Howto RL-HT-002: Optuna
- (c) Howto RL-ATT-001: Stagnation Detection

## 7.3 MLPro-GT-Native - Native Games

### 7.3.1 Player, Coalition, Competition

In game theory, the concepts of player, coalition, and competition are fundamental to understanding and analyzing strategic interactions. Here's a brief explanation of each term:

**(1) Player:** A player is an individual or entity that is involved in a strategic interaction within a game. Players can be individuals, businesses, countries, or any decision-making entities that have a set of possible actions or strategies from which to choose. Each player's goal is typically to maximize their own utility or payoff, taking into account the choices of others.

**(2) Coalition:** A coalition is a group of players who join together and coordinate their strategies to achieve common objectives. In cooperative game theory, players can form coalitions to enhance their collective outcomes. The formation and stability of coalitions are crucial aspects in understanding how players may collaborate or compete within a game. The concept is often used in the context of cooperative games, where players can benefit from working together.

**(3) Competition:** Competition in game theory refers to the rivalry or conflict among players as they pursue their individual goals. Games can involve varying degrees of competition, ranging from non-cooperative games where players act independently and competitively to cooperative games where players may form alliances to enhance their outcomes. Understanding the nature and intensity of competition is essential for predicting player behavior and outcomes in strategic interactions.

These concepts are integral to the analysis of different types of games, helping to model and predict the behavior of decision-makers in various situations.

#### Player

- Setup a Player

Setting up a player for a game in MLPro-GT-Native is very simple, as following:

```
from mlpro.gt.native.basics import *
from mlpro.gt.pool.native.solvers.randomsolver import RandomSolver #_
↳Optional

class MyGame(GTGame):

    C_NAME = 'MyGame'

    def _setup(self, p_mode, p_ada:bool, p_visualize:bool, p_logging) ->_
↳Model:

        # Setup strategy space
        _strategy_space = MSpace()
        _strategy_space.add_dim(Dimension('RStr', 'Z', 'Random Strategy', '', '
↳', '', [0,1]))

        # Setup a solver, e.g. Random Solver
        solver1 = RandomSolver(
            p_strategy_space=_strategy_space,
            p_id=1,
            p_name="Random Solver",
```

(continues on next page)

(continued from previous page)

```

        p_visualize=p_visualize,
        p_logging=p_logging
    )

    # Setup a player
    p1 = GTPlayer(
        p_solver=solver1,
        p_name="Player of Prisoner 1",
        p_visualize=p_visualize,
        p_logging=p_logging,
        p_random_solver=False
    )

    ...

```

- **Prerequisites**

- To set up a game, please refer to *games page*.
- To set up a solver, please refer to *solvers page*.

## Coalition

- **Setup a Coalition**

Setting up a colation for a game in MLPro-GT-Native is very simple, as following:

```

from mlpro.gt.native.basics import *

class MyGame(GTGame):

    C_NAME = 'MyGame'

    def _setup(self, p_mode, p_ada:bool, p_visualize:bool, p_logging) -> Model:

        # Setup strategy space
        ...

        # Setup a solver, e.g. Random Solver
        ...

        # Setup a set of players
        p1 = GTPlayer(...)
        p2 = GTPlayer(...)
        ...

        # Setup a coalition
        coal1 = GTCoalition(
            p_name="Coalition 1",
            p_coalition_type=GTCoalition.C_COALITION_SUM
        )
        coal1.add_player(p1)

```

(continues on next page)

(continued from previous page)

```
coal1.add_player(p2)
...
```

- **Prerequisites**

- To set up a game, please refer to *games page*.
- To set up a solver, please refer to *solvers page*.
- To set up a player, please refer to *setup player page*.

## Competition

- **Setup a Competition**

Setting up a competition for a game in MLPro-GT-Native is very simple, as following:

```
from mlpro.gt.native.basics import *

class MyGame(GTGame):

    C_NAME = 'MyGame'

    def _setup(self, p_mode, p_ada:bool, p_visualize:bool, p_logging) -> Model:

        # Setup strategy space
        ...

        # Setup a solver, e.g. Random Solver
        ...

        # Setup a set of players
        ...

        # Setup a set of coalitions
        coal1 = GTCoalition(...)
        coal2 = GTCoalition(...)
        ...

        # Setup a competition
        competition = GTCompetition(
            p_name="Prisoner's Dilemma Competition",
            p_logging=p_logging
        )
        competition.add_coalition(coal1)
        competition.add_coalition(coal2)
        ...

        ...
```

If you only have a competition between individual players, you can simply set up a set of coalitions with one player in each coalition. This configuration also allows a competition between players and coalitions.

- **Prerequisites**

- To set up a game, please refer to [games page](#).
- To set up a solver, please refer to [solvers page](#).
- To set up a player, please refer to [setup player page](#).
- To set up a coalition, please refer to [setup coalition page](#).

### 7.3.2 Payoff

In game theory, a payoff represents the numerical outcome or value that a player receives based on the combination of strategies chosen by all players in a given game. Payoffs are used to quantify the benefits or costs associated with different combinations of actions taken by the players.

There are two common ways to represent payoffs in game theory, which is also provided in MLPro-GT-Native:

**(1) Payoff Matrix:** A payoff matrix is a table that shows the payoffs for each player corresponding to all possible combinations of strategies. In a two-player game, the matrix typically has rows representing the strategies of one player and columns representing the strategies of the other player. The intersection of a row and a column provides the payoffs to each player based on their chosen strategies. Payoffs can be expressed as numerical values, utility units, or any other relevant measure.

**(2) Transfer Function:** In some cases, especially in cooperative game theory, payoffs may be expressed using a transfer function. A transfer function describes how the total value or utility generated by a coalition of players is distributed among its members. It specifies how the payoff gained by the coalition is divided among individual players. Transfer functions are often used in cooperative games to model scenarios where players form alliances and share the benefits of their joint actions.

In summary, payoffs in game theory are the numerical outcomes associated with different combinations of strategies chosen by players. Whether represented in a payoff matrix or through transfer functions, payoffs provide a quantitative measure of the success or utility that each player obtains in a given strategic interaction.

### Payoff Matrix

- **Setup a Payoff Matrix**

Setting up a payoff matrix for a game in MLPro-GT-Native is very simple. There are three main steps, as following:

```
from mlpro.gt.native.basics import *

class MyPayoff(GTFunction):

    def _setup_mapping_matrix(self) -> np.ndarray:
        # Step 1
        ...

    def _setup_payoff_matrix(self):
        # Step 2
        ...

class MyGame(GTGame):

    C_NAME = 'MyGame'
```

(continues on next page)

(continued from previous page)

```

def _setup(self, p_mode, p_ada:bool, p_visualize:bool, p_logging) ->
↳Model:

...

# Step 3
self._payoff = GTPayoffMatrix(
    p_function=MyPayoff(
        p_func_type=GTFunction.C_FUNC_PAYOFF_MATRIX,
        ...
    ),
    p_player_ids=coal_ids
)

...

```

- **Prerequisites**

- To set up a game, please refer to [games page](#).
- To set up players, coalitions, and a competition, please refer to [player, coalition, competition page](#).

## Transfer Function

- **Setup a Transfer Function for Payoffs**

Setting up a transfer function for payoffs in a game in MLPro-GT-Native is very simple. There are four main steps, as following:

```

from mlpro.gt.native.basics import *
from mlpro.bf.physics.basics import *

class MyTransferFunction(TransferFunction):

    def _set_function_parameters(self, p_args) -> bool:
        # Step 1.1
        ...

    def _custom_function(self, p_input, p_range=None):
        # Step 1.2 (Optional)
        ...

class MyPayoff(GTFunction):

    def _setup_transfer_functions(self):
        # Step 2
        TF = MyTransferFunction(...)

        self._add_transfer_function(p_idx=0, p_transfer_fct=TF)
        self._add_transfer_function(p_idx=1, p_transfer_fct=TF)
        ...

```

(continues on next page)

(continued from previous page)

```

class MyPayoffMatrix(GTPayoffMatrix):

    def _call_mapping(self, p_input:str, p_strategies:GTStrategy) -> float:
        # Step 3.1
        ...

    def _call_best_response(self, p_element_id:str) -> float:
        # Step 3.2
        ...

    def _call_zero_sum(self) -> bool:
        # Step 3.3
        ...

class MyGame(GTGame):

    C_NAME = 'MyGame'

    def _setup(self, p_mode, p_ada:bool, p_visualize:bool, p_logging) ->
↳Model:

        ...

        # Step 4
        self._payoff = MyPayoffMatrix(
            p_function=MyPayoff(
                p_func_type=GTFunction.C_FUNC_TRANSFER_FCTS
            ),
            p_player_ids=coal_ids
        )

        ...

```

- **Prerequisites**

- To set up a transfer function, please refer to *transfer function page*.
- To set up a game, please refer to *games page*.
- To set up players, coalitions, and a competition, please refer to *player, coalition, competition page*.

### 7.3.3 Solvers

In the context of game theory, solvers refer to computational tools or algorithms designed to find solutions to games. These solutions can include Nash equilibria, strategies, or other relevant outcomes that help analyze and understand the strategic interactions between players. Game theory solvers are particularly useful for complex games where manual analysis may be impractical or time-consuming. There are different types of solvers in game theory, depending on the nature of the game and the specific problem being addressed.

In MLPro-GT-Native, there are two possibilities to utilize a solver, either create a custom solver or reuse a solver from pool of solvers, as follows:



## Custom Solvers

- Setup a Custom Solver

Setting up a custom solver in MLPro-GT-Native is very simple. There are one method to be defined and other two methods to be optionally defined, as following:

```
from mlpro.gt.native.basics import *

class MySolver(GTSolver):

    C_NAME      = 'MySolver'

    # Optional
    def _init_hyperparam(self, **p_param):
        """
        Implementation specific hyperparameters can be added here.
        ↪ Please follow these steps:
            a) Add each hyperparameter as an object of type HyperParam
            ↪ to the internal hyperparameter
               space object self._hyperparam_space
            b) Create hyperparameter tuple and bind to self._hyperparam_
            ↪ tuple
            c) Set default value for each hyperparameter

        Parameters
        -----
        p_param : Dict
            Further model specific hyperparameters, that are passed
            ↪ through constructor.
        """

        ...

    # Optional
    def _setup_solver(self):
        """
        A method to setup a solver. This needs to be redefined
        ↪ based on each policy, but remains
           optional.
        """

        ...

    # Mandatory
    def _compute_strategy(self, p_payoff:GTPayoffMatrix) ->
    ↪ GTStrategy:
        """
        A method to compute a strategy from the solver. This
        ↪ method needs to be redefined.

        Parameters
```

(continues on next page)

(continued from previous page)

```
-----  
p_payoff : GTPayoffMatrix  
    Payoff matrix of a specific player.  
  
Returns  
-----  
GTStrategy  
    The computed strategy.  
  
"""  
  
...
```

- **Algorithm Checker**

A test script using a unit test to check the developed solvers will be available soon!

## Solvers Pool

### Random Solvers

#### General Information

A random solver for a specific strategy space with defined boundaries is an algorithm that generates random values within a specific range or set of constraints. This type of random solver is often used for testing environments, generate random behaviour of competitors or alliances, and many more. The random generator can be attach to an GT player/coalition and will detect the strategy space of the player/coalition as well as the boundaries. The strategy space refers to the set of possible strategies of the agent, and the boundaries define the range of values within which the strategy must lie. Then, a strategy is randomly computed by the generator using uniform distribution.

The random solvers can be imported via:

```
from mlpro.gt.pool.native.solvers.randomsolver import RandomSolver
```

#### Cross Reference

- *Howto GT-Native-001: 2P Prisoners Dilemma*
- *API Reference*

#### Citation

If you apply this solver in your research or work, please *[cite](#)* us.

## Min-Max Greedy Solvers

### General Information

In game theory, the minimum and maximum greedy policy is a decision-making strategy used by players to optimize their outcomes in zero-sum or even general-sum games. Zero-sum games are situations where one player's gain or loss is exactly balanced by the losses or gains of other players, while general-sum games are the opposite. The term “greedy” refers to the strategy of maximizing one's own payoff while minimizing the potential payoff of the opponent.

The maximum greedy can be imported via:

```
from mlpro.gt.pool.native.solvers.greedypolicy import MaxGreedyPolicy
```

The minimum greedy can be imported via:

```
from mlpro.gt.pool.native.solvers.greedypolicy import MinGreedyPolicy
```

### Cross Reference

- [Howto GT-Native-002: 3P Prisoners Dilemma](#)
- [Howto GT-Native-005: 3P Routing Problems](#)
- [API Reference](#)

### Citation

If you apply this solver in your research or work, please [cite](#) us.

## 7.3.4 Games

A “game” refers to a formalized model of strategic interaction among rational decision-makers, known as players. These players can be individuals, companies, countries, or any entities capable of making decisions. Games in game theory are designed to analyze situations where the outcome for each player depends not only on their own actions but also on the actions of others. Key components of a game in game theory include players, strategies, payoffs, and more.

Games can be classified into various types based on different criteria, such as the number of players, the information available to players, the timing of moves, and the degree of cooperation among players. Common types of games include simultaneous games, sequential games, cooperative games, non-cooperative games, and more. The objective of game theory is to analyze and predict the strategic interactions and outcomes in these games, providing insights into decision-making, cooperation, competition, and rational behavior in a wide range of fields.

In MLPro-GT-Native, there are two possibilities to start a game, either create a custom game or reuse a game from pool of games, as follows:

### Custom Games

- **Prerequisites**
  - To set up a player, coalition, and competition, please refer to [player, coalition, and competition page](#).
  - To set up a payoff function or matrix, please refer to [payoff page](#).
  - To set up a solver, please refer to [solvers page](#).
- **Setup a Custom Game**

If you already know how to set up coalition/competition, payoff function/matrix, and solver. Then, setting up a custom game in MLPro-GT-Native is very simple, where you only need to bring everything together in one method, as follows:

```
from mlpro.gt.native.basics import *

class MyGame(GTGame):

    C_NAME = 'MyGame'

    def _setup(self, p_mode, p_ada:bool, p_visualize:bool, p_
    logging) -> Model:

        # Setup strategy space
        ...

        # Setup a solver, e.g. Random Solver
        ...

        # Setup a set of players
        ...

        # Setup a set of coalitions
        ...

        # Setup a competition (if required)
        ...

        # Setup the payoff matrix or function
        self._payoff = ...

    return Model
```

## Games Pool

### 2P Prisoners' Dilemma

#### General Information

The Prisoner's Dilemma is a classic concept in game theory that describes a situation where two players must decide whether to cooperate or betray each other, with the outcomes determined by the combined choices made by both players. In a two-player Prisoner's Dilemma game, the players face a payoff matrix showing possible outcomes and associated payoffs for each player based on their choices.

Each player has two possible choices: "Confess" or "Not Confess". The players make their decisions independently without knowing the other player's choice. The payoffs for each player depend on both their choice and the choice made by the other player, as outlined in the payoff matrix.

The four possible outcomes in a two-player Prisoner's Dilemma game are (Confess, Confess), (Confess, Not Confess), (Not Confess, Confess), and (Not Confess, Not Confess). The dilemma arises because, individually, each player has an incentive to betray the other to achieve a higher payoff, regardless of the other player's choice.

The Nash equilibrium in this game is typically the outcome where both players betray each other, as neither player can unilaterally change their strategy to improve their payoff. However, from a collective perspec-

tive, both players cooperating would lead to a better overall outcome. The tension lies in the fact that betraying is the dominant strategy for each player, even though the jointly optimal outcome would be cooperation.

The Prisoner's Dilemma is used to illustrate the challenges of cooperation in situations where individual incentives may lead to suboptimal collective outcomes. It has applications in various fields to analyze situations involving cooperation and competition.

This game can be imported, as follows:

```
from mlpro.gt.pool.native.games.prisonersdilemma_2p import _
↳ PrisonersDilemma2PGame
```

### Player, Coalition, and Competition

In the context of the Prisoner's Dilemma, there are two players, often referred to as Player 1 and Player 2. The situation involves two individuals who have been accused of committing a crime together and are now being held in separate cells, unable to communicate with each other. The choices available to each player are either to "Confess" or "Not Confess". The outcomes and associated payoffs for each player depend on the combination of choices made by both individuals.

### Payoff Matrix

	Player 2: Confess	Player 2: Not Confess
Player 1: Confess	(2, 2)	(8, 1)
Player 2: Not Confess	(1, 8)	(5, 5)

### Solvers

Player	Solvers
1	Random Solver
2	Random Solver

### Cross References

- [Howto GT-Native-001: 2P Prisoners Dilemma](#)
- [API Reference](#)

### Citation

If you apply this game in your research or work, do not forget to [cite us](#).

## 3P Prisoners' Dilemma

### General Information

The Prisoner's Dilemma can be extended to involve three players, creating what is known as a 3-player Prisoner's Dilemma game. In this scenario, three individuals face a situation where they must decide whether to cooperate or betray each other, with the outcomes determined by the combined choices made by all three players. The setup involves a payoff matrix that outlines the possible outcomes and associated payoffs for each player based on their choices.

Each player has two possible choices: "Confess" or "Not Confess". The outcomes are determined based on the combination of choices made by all three players. The challenge is that, similarly to the 2-player version, there is a conflict between individual rationality and collective optimality.

The players individually have an incentive to not confess, as not confessing typically yields a higher payoff for the individual, regardless of the choices made by others. However, if all players not confessing, the collective outcome may be suboptimal compared to a scenario where all players cooperate.

Analyzing and solving a 3-player Prisoner's Dilemma involves considering the strategic interactions among all three players, understanding the incentives for cooperation and betrayal, and exploring whether there are any stable strategies or equilibria in the game. The extension to more players adds complexity to the decision-making dynamics and strategic considerations.

This game can be imported, as follows:

```
from mlpro.gt.pool.native.games.prisonersdilemma_3p import _
↪PrisonersDilemma3PGame
```

### Player, Coalition, and Competition

In the context of the Prisoner's Dilemma, there are three players, often referred to as Player 1, Player 2, and Player 3. The situation involves three individuals who have been accused of committing a crime together and are now being held in separate cells, unable to communicate with each other. The choices available to each player are either to "Confess" or "Not Confess". The outcomes and associated payoffs for each player depend on the combination of choices made by all individuals.

### Payoff Matrix

(P1, P2, P3)	P2: Confess, P3: Confess	P2: Confess, P3: Not Confess	P2: Not Confess, P3: Confess	P2: Not Confess, P3: Not Confess
P1: Confess	(2, 2, 2)	(5, 5, 1)	(5, 1, 5)	(10, 1, 1)
P2: Not Confess	(1, 5, 5)	(1, 10, 1)	(1, 1, 10)	(15, 15, 15)

### Solvers

Player	Solvers
1	Random Solver, Min Greedy Solver
2	Min Greedy Solver
3	Random Solver, Min Greedy Solver

### Cross References

- [Howto GT-Native-002: 3P Prisoners Dilemma](#)
- [API Reference](#)

### Citation

If you apply this game in your research or work, do not forget to [cite us](#).

## Rock, Paper, Scissors

### General Information

Rock, Paper, Scissors is a simple hand game often used as a playful decision-making tool. In game theory, it serves as a basic example of a non-cooperative, simultaneous-move, zero-sum game. The game is typically played between two people who simultaneously form one of three shapes with their hands: a rock, paper, or scissors.

The basic rules are as follows:

- Rock crushes Scissors,
- Scissors cuts Paper, and
- Paper covers Rock.

The game has a cyclic nature, with no single option dominating the others. Each choice has a clear advantage over one option and a disadvantage against another.

The game is designed such that the payoffs balance out to zero in each possible outcome, making it a zero-sum game. In a theoretical sense, neither player has a dominant strategy, and the optimal strategy involves randomizing between Rock, Paper, and Scissors to prevent the opponent from predicting their moves.

Rock, Paper, Scissors is often used as a teaching tool in introductory game theory to illustrate concepts such as mixed strategies, Nash equilibrium, and the nature of zero-sum games. Despite its simplicity, it provides insights into strategic thinking and decision-making in competitive situations.

This game can be imported, as follows:

```
from mlpro.gt.pool.native.games.rockpaperscissors import RockPaperScissors
```

### Player, Coalition, and Competition

In the context of Rock, Paper, Scissors, there are usually two players, often referred to as Player 1 and Player 2. However, in this game, we setup a coalition against a coalition Rock, Paper, Scissors games. The game consists of two coalitions, where each coalition makes a decision based on the collaborative approach between the coalitions. Each coalition consists of 5 members, the most voted decision of the 5 members represents the final decision of the coalition.

### Payoff Matrix

The outcomes are often represented in a payoff matrix, where each coalition's payoff depends on the combination of choices made by both coalitions:

	Rock	Paper	Scissors
Rock	(0, 0)	(0, 1)	(1, 0)
Paper	(1, 0)	(0, 0)	(0, 1)
Scissors	(0, 1)	(1, 0)	(0, 0)

Here, the first value in each pair represents the payoff to the row coalition, and the second value represents the payoff to the column coalition.

### Solvers

Coalition	Solvers
1	Random solvers for all players
2	Random solvers for all players

### Cross References

- [Howto GT-Native-003: Rock, Paper, Scissors](#)
- [API Reference](#)

### Citation

If you apply this game in your research or work, do not forget to [cite us](#).

## 3P Supply and Demand

### General Information

In a 3-seller supply and demand game within the framework of game theory, three sellers, denoted as Seller 1, Seller 2, and Seller 3, engage in strategic decision-making alongside a group of buyers in a market where goods or services are exchanged. Each seller must decide on the quantity of goods they wish to supply/produce to the market, while buyers independently determine the quantity they want to demand. The payoff for sellers is typically associated with their revenue, contingent on the quantity they sell and the price they set, while buyers' payoffs may relate to the utility they derive from purchased goods and the corresponding prices.

The interaction between the supply decisions of sellers and the demand decisions of buyers is crucial in determining the market dynamics. Sellers strive to maximize their revenue, considering factors like quantity and pricing, while buyers aim to maximize their utility by deciding on the quantity they wish to purchase and the price they are willing to pay. The equilibrium, a fundamental concept in competitive markets, occurs when the quantity supplied equals the quantity demanded, establishing a market-clearing price.

Strategic considerations play a pivotal role as sellers may take into account the decisions of other sellers when determining their own supply/production quantities and pricing strategies. Similarly, buyers may consider the prices set by sellers and the available quantities when deciding on their demand quantities. The Nash equilibrium, a key concept in game theory, is reached when each participant's strategy is optimal given the strategies chosen by others.

In this market setting, competition arises as sellers may compete against each other to attract buyers, adjusting prices or quantities to gain a competitive edge. Alternatively, sellers may engage in strategic co-operation, such as forming a cartel, to collectively influence prices and quantities in the market. The game dynamics involve repeated interactions between sellers and buyers, leading to adjustments in strategies based on past outcomes and market conditions.

Analyzing this type of game requires an examination of the strategies and payoffs for both sellers and buyers, identification of potential equilibria, and an understanding of how the strategic decisions of each participant influence the overall market dynamics. Game theory provides a valuable framework for studying the strategic interactions and decision-making processes in complex market scenarios involving multiple sellers and buyers.

This game can be imported, as follows:

```
from mlpro.gt.pool.native.games.supplydemand_3p import SupplyDemand_3P
```

### Player, Coalition, and Competition

In the 3P Supply and Demand games in MLPro-GT, we set up three players with a constant market demand of the products is 10 products/day, where the buyer will always firstly buy the products with less prices. Each player in the game represent each seller. They need to compete or cooperate with each other to maximize their individual profit.

### Payoff Function



The table below shows the sales price per product based on the quantity produced by each seller:

Seller 1		Seller 2		Seller 3	
Price (€)	Quantity	Price (€)	Quantity	Price (€)	Quantity
15	1	10	1	8	1
12	2	8	2	7	2
9	3	6	3	6	3
6	4	4	4	5	4
3	5	2	5	4	5

### Solvers

Seller	Solvers
1	Max Greedy Solver
2	Max Greedy Solver
3	Random Solver

### Cross References

- [Howto GT-Native-004: 3P Supply and Demand](#)
- [API Reference](#)

### Citation

If you apply this game in your research or work, do not forget to [cite us](#).

## 3P Routing Problems

### General Information

In a 3-player routing problem with congestion in game theory, three entities or players navigate a network with multiple routes to reach their respective destinations. Each player's objective is to minimize their travel time or cost, and the congestion on each route is a critical factor influenced by the collective decisions of all players. This scenario is formalized as a congestion game, a type of non-cooperative game where individual players make decentralized decisions impacting overall congestion levels.

The network comprises various routes, each associated with a travel time or cost. Importantly, the congestion on a route increases as more players opt for that specific path, leading to longer travel times or higher costs. Players make independent decisions regarding their routes, aiming to optimize their individual utility based on minimizing travel time or cost. The strategic considerations involve players weighing the congestion effects on different routes when making their decisions.

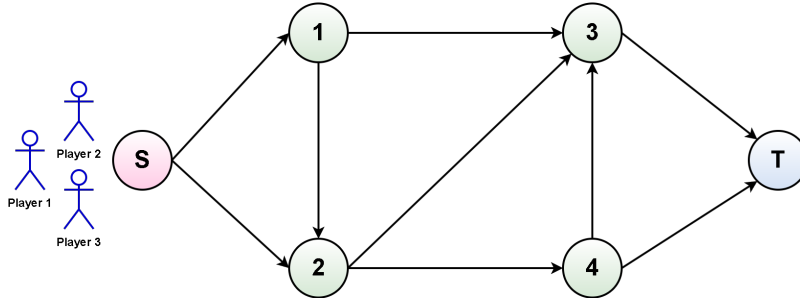
A Nash equilibrium, a key concept in game theory, is reached when no player has an incentive to unilaterally change their chosen route given the choices made by others. At equilibrium, the congestion on each route stabilizes, and players find routes that collectively satisfy their individual optimization goals.

In this setting, players engage in strategic decision-making by considering the congestion levels on different routes. They evaluate the trade-off between selecting a potentially faster yet more congested route against a longer but less congested alternative. The social welfare of the system, reflecting overall efficiency, is influenced by the aggregate travel times or costs across all players and routes.

Analyzing 3-player routing problems with congestion involves exploring the strategic interactions among players, identifying potential Nash equilibria, and assessing the implications of individual decisions on the overall efficiency of the network. Game theory serves as a valuable framework for understanding how

decentralized decision-making in such scenarios can lead to equilibrium outcomes and how those outcomes impact network congestion dynamics.

In this game, we setup 3P routing problems with congestions, as follows:



The goal is to reach the target node from the starting node with the shortest pathway as possible and avoid congestions as much as possible.

This game can be imported, as follows:

```
from mlpro.gt.pool.native.games.routingproblems_3p import Routing_3P
```

### Player, Coalition, and Competition

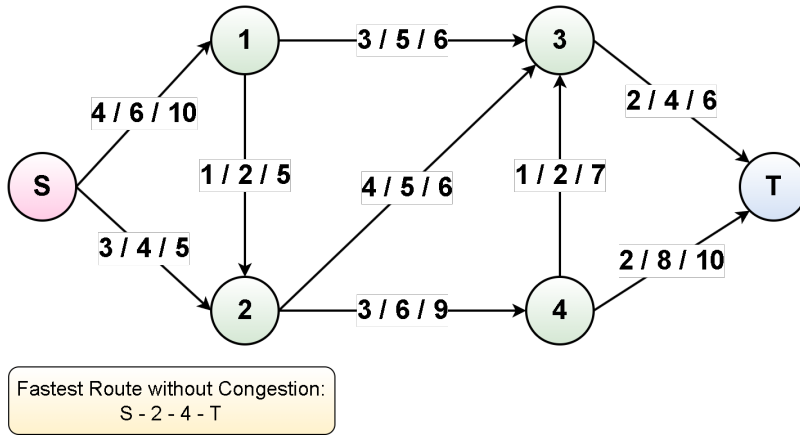
In the 3P Routing Problems games in MLPro-GT, we consider three players. Each player has a set of strategies to be selected, as follows:

Strategy	Path
1	S -> 1 -> 2 -> 3 -> T
2	S -> 1 -> 3 -> T
3	S -> 1 -> 2 -> 4 -> 3 -> T
4	S -> 1 -> 2 -> 4 -> T
5	S -> 2 -> 4 -> T
6	S -> 2 -> 4 -> 3 -> T
7	S -> 2 -> 3 -> T

### Payoff Function

The figure below shows the traveling time from one node to another, where each path includes potentially three distances (X / Y / Z).

- X = the travelling time, if only one player chooses this path
- Y = the travelling time, if two players choose this path simultaneously
- Z = the travelling time, if three players choose this path simultaneously



### Solvers

Player	Solvers
1	Min Greedy Solver
2	Min Greedy Solver, Random Solver
3	Random Solver

### Cross References

- [Howto GT-Native-005: 3P Routing Problems](#)
- [API Reference](#)

### Citation

If you apply this game in your research or work, do not forget to [cite us](#).

## 7.4 MLPro-GT-DG - Dynamic Games

### 7.4.1 Game Boards

A game board is a “place” where players perform and interact to each other in game theory. There are three main possibilities to set up a game board in MLPro-GT-DG, such as:

- (1) the user can develop a custom game board,
- (2) the user can reuse available RL Environments from MLPro, OpenAI Gym, or PettingZoo, and
- (3) the user can reuse the provided game boards by accessing them from the pool of objects.



## Custom Game boards

### MLPro GameBoard Model

- Latency defines the time needed for the game board to react to the input.
- Supports single and multi-player control (see C\_TYPE\_EVERY\_AGENT).
- Supports simulation and real control mode.
- Wrappers for OpenAI Gym and Petting Zoo available.
- Because of inheritance an GameBoard object can also be treated as a single Reward/Done/Broken function.

**Hint for developers:** only the **blue** constants, attributes and methods need to be implemented.

### Setup Steps:

There are plenty of methods present, but by following the green bubbles, the game board will be set up and be ready to use.

### Set Item Methods:

- `_setup_spaces()` needs to be implemented to enrich the state and action space with specific dimensions.
- The random seed, latency, and mode of the game board can be set explicitly by calling the respective functions.

### Action Processing Methods:

- When the mode is set to Real, the `process_action` method will forward the Action input to `_export_action` and wait for the feedback received from `_import_state`.
- When the mode is set to Sim, the `process_action` method will forward the Action input to `simulate_reaction` and store the new state using `_set_state`.
- The `process_action` method then continues by computing the reward, done, broken and goal achievement implemented in the respective functions.
- The reset method should reset the game board to initial state.

### Get Item Methods:

- Mode defines whether the game board is Simulated or Real.
- Cycle Limit defines the limit for training episodes.

**Hint for developers:** The following methods will return the respective information.

### Logging Methods:

- Logging can be switched using `switch_logging` method
- The items can be logged manually using the `log` method

### Plotting Methods:

- The plotting area should be initialized in the `init_plot` method.
- The `update_plot` method should be implemented so that the plotting area is updated.

```

class GameBoard(Environment)
    C_TYPE = 'Game Board'
    C_NAME = '????'
    C_CYCLE_LIMIT = 0
    C_LATENCY = timedelta(0,1,0)
    C_REWARD_TYPE = Reward.C_TYPE_EVERY_AGENT

    __init__( p_mode=Mode.C_MODE_SIM,
              p_latency=None,
              p_afct_strans=None,
              p_afct_reward=None,
              p_afct_success=None,
              p_afct_broken=None,
              p_logging=Log.C_LOG_ALL )

    STATIC setup_spaces(): MSpace, MSpace
    set_random_seed( p_seed=None )
    set_latency( p_latency:timedelta=None )
    set_mode( p_mode )
    reset( p_seed=None )
    process_action( p_action:Action ): bool
    _set_state(p_state)
    simulate_reaction( p_state:State, p_action:Action ): State
    compute_reward( p_state_old:State,
                    p_state_new:State ): Reward
    compute_success( p_state:State ): bool
    compute_broken( p_state:State ): bool
    clear_buffer()
    _reset( p_seed=None )
    _process_action( p_action:Action ): bool
    _simulate_reaction( p_state:State, p_action:Action ): State
    _compute_broken( p_state:State ): bool
    _compute_success( p_state:State ): bool
    _compute_reward( p_state_old:State,
                     p_state_new:State ): Reward
    _export_action(p_action)
    _import_state()
    _utility_fct(p_state:State, p_player_id)

    get_mode()
    get_latency()
    get_cycle_limit()
    get_action_space()
    get_state_space()
    get_state()
    get_reward_type()
    get_success()
    get_broken()
    get_last_reward()
    get_functions(): AFctSTrans, AFctReward, AFctSuccess,
                    AFctBroken
    switch_logging(p_logging=True)
    log(p_type, *p_args)
    init_plot(p_figure=None)
    update_plot()
  
```

1. Create your own class and inherit this class.

2. Setup state and action space here.

3. Add the simulation code here or inside an AdaptiveFunction

4. If you want to control real hardware, just implement these two methods

5. The game board is ready to be paired with a player inside a scenario!

- Game Board Creation for Simulation Mode

Creating a game board that satisfies the MLPro interface is immensely simple and straightforward. Basically, an MLPro game board is a class with several main functions. Each game board must apply the following MLPro functions:

```
from mlpro.gt.models import *

class MyGameBoard(GameBoard):
    """
    Custom game board that satisfies mlpro interface.
    """
    C_NAME          = 'MyGameboard'
    C_LATENCY        = timedelta(0,1,0)          # Default latency 1s
    C_REWARD_TYPE    = Reward.C_TYPE_EVERY_AGENT # Default reward type

    def __init__(self, p_mode=C_MODE_SIM, p_latency:timedelta=None, p_
↪ logging=True):
        """
        Parameters:
            p_mode          Mode of environment (simulation/real)
            p_latency        Optional: latency of environment. If not_
↪ provided
                            internal value C_LATENCY will be used by_
↪ default
            p_logging       Boolean switch for logging
        """
        super().__init__(p_latency=p_latency, p_logging=p_logging)
        self._setup_spaces()
        self.set_mode(p_mode)

    def _setup_spaces(self):
        """
        Static template method to set up and return state and action space_
↪ of environment.

        Returns
        -----
        state_space : MSpace
            State space object
        action_space : MSpace
            Action space object
        """
        # Setup state space example
        # self.state_space.add_dim(Dimension('Pos', 'Position', ", 'm', 'm', [-
↪ 50,50]))
        # self.state_space.add_dim(Dimension('Vel', 'Velocity', ", 'm/sec', '\
↪ frac{m}{sec}', [-50,50]))

        # Setup action space example
        # self.action_space.add_dim(Dimension('Rot', 'Rotation', ", '1/sec', '\
↪ frac{1}{sec}', [-50,50]))
```

(continues on next page)

(continued from previous page)

```

    ....

    def _process_action(self, p_action: Action) -> bool:
        """
        Custom method for state transition. To be implemented in a child
        ↪class. See method
        process_action() for further details.

        Parameters
        -----
        p_action : Action
            Action to be processed

        Returns
        -----
        success : bool
            True, if action processing was successfull. False otherwise.
        """
        ....

    def _simulate_reaction(self, p_state: State, p_action: Action) -> State:
        """
        Custom implementation to simulate a state transition. See method
        ↪simulate_reaction() for
        further details.

        Parameters
        -----
        p_state : State
            Current state.
        p_action : Action
            Action.

        Returns
        -----
        State
            Subsequent state after transition
        """
        ....

    def _reset(self, p_seed=None) -> None:
        """
        Custom method to reset the environment to an initial/defined state.

        Parameters
        -----
        p_seed : int
            Seed parameter for an internal random generator

        """
        ....

```

(continues on next page)

(continued from previous page)

```

def _compute_reward(self, p_state_old: State, p_state_new: State) ->
↳Reward:
    """
    Custom reward computation method. See method compute_reward() for
↳further details.

    Parameters
    -----
    p_state_old : State
        Optional state before transition. If None the internal previous
↳state of the environment
        is used.
    p_state_new : State
        Optional tate after transition. If None the internal current
↳state of the environment
        is used.

    Returns
    -----
    Reward
        Reward object.
    """
    . . .

def _compute_success(self, p_state: State) -> bool:
    """
    Custom method for state evaluation 'success'. See method compute_
↳success() for further details.

    Parameters
    -----
    p_state : State
        State to be assessed.

    Returns
    -----
    bool
        True, if the given state is a 'success' state. False otherwise.
    """
    . . .

def _compute_broken(self, p_state: State) -> bool:
    """
    Custom method for state evaluation 'broken'. See method compute_
↳broken() for further details.

    Parameters
    -----
    p_state : State
        State to be assessed.

    Returns
    -----

```

(continues on next page)



(continued from previous page)

```

-----
bool
    True, if the given state is a 'broken' state. False otherwise.
    """
    ....

def _utility_fct(self, p_state: State, p_player_id):
    """
    Computes utility of given player. To be redefined.
    """
    ....

```

One of the benefits for MLPro users is the variety of reward structures, which is useful for a multi-player GT approach. Three types of reward structures are supported in this framework, such as:

1. **C\_TYPE\_OVERALL** is the default type and is a scalar overall value,
2. **C\_TYPE EVERY\_AGENT** is a scalar for every player, and
3. **C\_TYPE EVERY\_ACTION** is a scalar for every player and action.

To set up state- and action spaces using our basic functionalities, please refer to *Howto GT-002: Train Multi-Player*.

#### • Game Board Creation for Real Hardware Mode

In MLPro, we can choose simulation mode or real hardware mode. For real hardware mode, the creation of an environment is very similar to the simulation mode. You do not need to define **\_simulate\_reaction**, but you need to replace it with **\_export\_action** and **\_import\_state** as it is shown in the following:

```

def _export_action(self, p_action: Action) -> bool:
    """
    Mode C_MODE_REAL only: exports given action to be processed externally
    (for instance by a real hardware). Please redefine.

    Parameters
    -----
    p_action : Action
        Action to be exported

    Returns
    -----
    bool
        True, if action export was successful. False otherwise.

    """

    raise NotImplementedError

def _import_state(self) -> bool:
    """
    Mode C_MODE_REAL only: imports state from an external system (for
    instance a real hardware).
    Please redefine. Please use method set_state() for internal update.

```

(continues on next page)

(continued from previous page)

```

Returns
-----
bool
    True, if state import was successful. False otherwise.

"""

raise NotImplementedError

```

- **Game Board Checker**

To check whether your developed game board is compatible with the MLPro interface, we provide a test script using Unittest. At the moment, you can find the source code [here](#). We will prepare a built-in testing module in MLPro, show you how to execute the testing soon and provides an example as well.

## Reusing RL Environments

- **Transferring RL Environment to GT Game Board**

In MLPro, we can simply transfer an RL environment to a GT game board by inheriting GameBoard functionality, as it is shown in the following:

```

from mlpro.gt.models import *
from mlpro.rl.pool.envs.dummy_environment import DummyEnv

class MyGameBoard_GT(DummyEnv, GameBoard):
    C_NAME = 'MyGameBoard_GT'

    def __init__(self, p_logging=True):
        DummyEnv.__init__(self, p_reward_type=Reward.C_TYPE_EVERY_AGENT)

```

- **Game board from Third Party Packages**

Alternatively, if your environment follows Gym or PettingZoo interface, you can apply our relevant useful wrappers for the integration between third-party packages and MLPro. For more information about the available third-party packages, please click [here](#). Then, you need to transfer the wrapped RL environment to a GT Game Board.

## Cross Reference

- [MLPro-RL](#)

## Game board Pool

### BGLP

This game board reuses a native implementation of RL environment, which you can find the details [here](#).

This BGLP game board can be imported via:

```
import mlpro.gt.pool.boards.bglp
```

## Multicartpole

This game board reuses a native implementation of RL environment, which you can find the details [here](#).

This multi-cartpole game board can be imported via:

```
import mlpro.gt.pool.boards.multicartpole
```

### 7.4.2 Players

In the game theoretical approach, the player takes decisions based on the actual states in a game. A game contains two or more players that can be working cooperatively or compete with each other. Each player is supplied with a policy, where the policy can be used for the decision-making process and also optimized. The decision-making of the player is in the form of the next action.

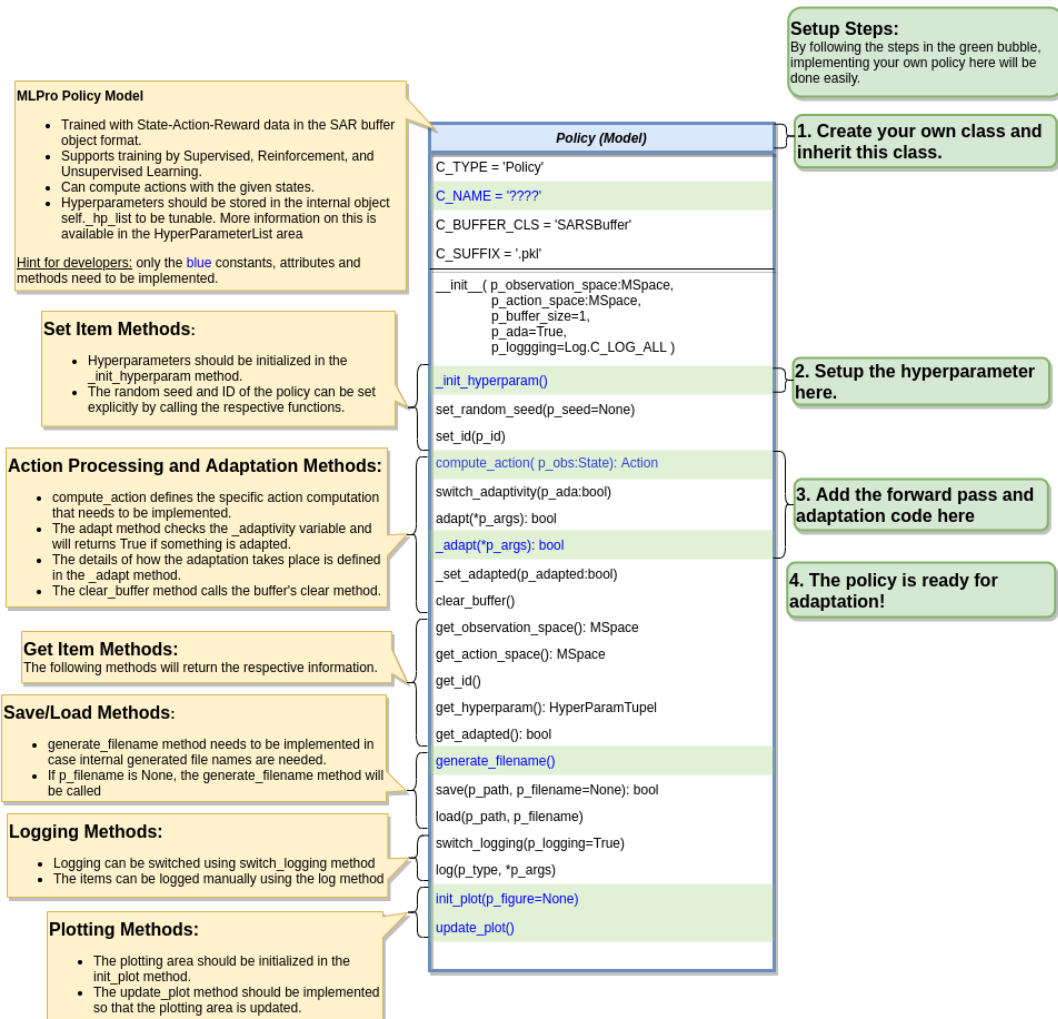
The three main tasks of the player are as follows:

- (1) Compute new action based on the current state.
- (2) Calculate local utility value based on the previous state and next state.
- (3) Optimize their policy based on the state and the selected action.

In MLPro, you can customize your GT-based policy or import the provided policies in the pool of objects.

#### Custom Policies

- Policy Creation



Creating a GT policy that satisfies the MLPro interface is pretty straightforward. You just require to assure that the GT policy consists of at least the following 3 main functions:

```

from mlpro.rl.models import *

class MyPolicy(Policy):
    """
    Creates a policy that satisfies the mlpro interface.
    """
    C_NAME = 'MyPolicy'

    def _init_hyperparam(self, **p_par):
        """
        Implementation-specific hyperparameters can be added here. Please
        follow these steps:
        a) Add each hyperparameter as an object of type HyperParam to the
        internal hyperparameter
        space object self._hyperparam_space
        b) Create a hyperparameter tuple and bind it to self._hyperparam_
        tuple
        c) Set the default value for each hyperparameter
  
```

(continues on next page)

(continued from previous page)

```

Parameters
-----
p_par : Dict
    Further model-specific parameters, that are passed through the
↳ constructor.

    """
    . . .

def compute_action(self, p_obs: State) -> Action:
    """
    Specific action computation method to be redefined.

    Parameters
    -----
    p_obs : State
        Observation data.

    Returns
    -----
    action : Action
        Action object.

    """
    . . .

def _adapt(self, *p_args) -> bool:
    """
    Adapts the policy based on State-Action-Reward-State (SARS) data.

    Parameters
    -----
    p_arg[0] : SARSElement
        Object of type SARSElement.

    Returns
    -----
    adapted : bool
        True, if something has been adapted. False otherwise.

    """
    . . .

```

To set up a hyperparameter space, please refer to *Howto BF-ML-010: Hyperparameters*.

- **Algorithm Checker**

A test script using the unit test to check the developed policies will be available soon!

## Players Pool

This will be available soon!

### 7.4.3 Games

Game theory encompasses various types of games, each with its unique characteristics and applications. Potential games, for example, are characterized by the existence of a potential function that simplifies the analysis of equilibrium strategies. In these games, players' best responses collectively minimize a global cost or potential.

Congestion games involve players choosing strategies in a shared environment where the payoff for each player depends on the congestion or competition for resources. Traffic networks and communication systems often serve as examples of congestion games.

Stackelberg games are a type of sequential game where one player, known as the leader, makes decisions first, and the other player, the follower, observes those decisions before making their own. This model is often applied in economic and business contexts, such as pricing and quantity decisions in a supply chain.

Other types of games include cooperative games where players form coalitions to achieve joint objectives and non-cooperative games like the prisoner's dilemma, where self-interest guides decision-making. Evolutionary games explore strategic interactions with a focus on the long-term evolution of strategies within a population.

Overall, the diversity of game types within game theory allows for a rich understanding of strategic behavior in various domains, providing valuable insights for decision-makers and analysts.

In the actual MLPro-GT-DG, we provide standardized template modules for two type of games, such as:

- (1) Potential Games
- (2) Stackelberg Games

#### Cross Reference

- *Howto GT-DG-003: Potential Games*
- *Howto GT-DG-004: Stackelberg Games*

## GENERAL INFORMATION

All extensions currently available in GitHub are listed in the MLPro extension hub.

### 8.1 What is an MLPro Extension?

An MLPro extension is technically a public GitHub repository that meets the following criteria:

- Topic *mlpro-extension*
- At least one release
- Recognizable reference to MLPro
- Documentation of all functionalities
- Executable and understandable example programs
- Terms of use/License
- Clean code (if provided)

### 8.2 How to contribute to the MLPro extension hub?

Placing your own MLPro extension here on the extension hub is free. Simply create and publish your extension as described above. It will be automatically detected, and a GitHub issue will be created for inclusion in the extension hub. This issue prompts us to review and incorporate your extension. All administrators of the extension are automatically involved in the comment area, where we can also discuss questions or annotations on demand. Once the review is done, your extension will be released to the extension hub. It will then automatically appear here in the documentation the following day.

We recommend using our template <https://github.com/fhswf/MLPro-Extension> to create your own extensions.

We warmly thank all contributors for your valued work!

## 8.3 Disclaimer

The respective third-party providers are responsible for the extensions listed here in the extension hub. This applies in particular to the quality, functionality, stability, and seriousness of the extensions. If you have any questions about an extension, please contact the responsible third-party provider directly.



## THIRD-PARTY EXTENSIONS

All MLPro extensions currently available in GitHub are listed below.

### 9.1 Organizations

This section lists MLPro extensions provided by organizations.

#### 9.1.1 Fachhochschule Südwestfalen

Wir geben Impulse!

<b>Location</b>	Iserlohn, Germany
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf">https://github.com/fhswf</a>
<b>URL</b>	<a href="http://fh-swf.de">http://fh-swf.de</a>

#### MLPro Extensions

##### MLPro-Int-Gymnasium

MLPro: Integration Gymnasium

<b>Topics</b>	gymnasium, machine-learning, mlpro-extension, reinforcement-learning, wrapper
<b>Version</b>	v1.0.1 - Bugfix
<b>Last update</b>	Sun, 21 Apr 2024 13:29:13 GMT
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf/MLPro-Int-Gymnasium">https://github.com/fhswf/MLPro-Int-Gymnasium</a>
<b>URL</b>	<a href="https://mlpro-int-gymnasium.readthedocs.io">https://mlpro-int-gymnasium.readthedocs.io</a>
<b>License</b>	Apache License 2.0

## MLPro-Int-Hyperopt

MLPro: Integration Hyperopt

<b>Topics</b>	hyperopt, hyperparameter-tuning, machine-learning, mlpro-extension, wrapper
<b>Version</b>	v1.0.0 - Hello World!
<b>Last update</b>	Thu, 18 Apr 2024 16:09:43 GMT
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf/MLPro-Int-Hyperopt">https://github.com/fhswf/MLPro-Int-Hyperopt</a>
<b>URL</b>	<a href="https://mlpro-int-hyperopt.readthedocs.io">https://mlpro-int-hyperopt.readthedocs.io</a>
<b>License</b>	Apache License 2.0

## MLPro-Int-OpenML

MLPro: Integration OpenML

<b>Topics</b>	machine-learning, mlpro-extension, openml, wrapper
<b>Version</b>	v1.0.0 - Hello World!
<b>Last update</b>	Thu, 18 Apr 2024 17:21:30 GMT
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf/MLPro-Int-OpenML">https://github.com/fhswf/MLPro-Int-OpenML</a>
<b>URL</b>	<a href="https://mlpro-int-openml.readthedocs.io">https://mlpro-int-openml.readthedocs.io</a>
<b>License</b>	Apache License 2.0

## MLPro-Int-Optuna

None

<b>Topics</b>	hyperparameter-tuning, machine-learning, mlpro-extension, optuna, wrapper
<b>Version</b>	v1.0.0 - Hello World!
<b>Last update</b>	Thu, 18 Apr 2024 16:54:13 GMT
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf/MLPro-Int-Optuna">https://github.com/fhswf/MLPro-Int-Optuna</a>
<b>URL</b>	<a href="http://mlpro-int-optuna.readthedocs.io/">http://mlpro-int-optuna.readthedocs.io/</a>
<b>License</b>	Apache License 2.0

## MLPro-Int-PettingZoo

MLPro: Integration PettingZoo

<b>Topics</b>	machine-learning, mlpro-extension, pettingzoo, reinforcement-learning, wrapper
<b>Version</b>	v1.0.0 - Hello World!
<b>Last update</b>	Fri, 19 Apr 2024 08:00:56 GMT
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf/MLPro-Int-PettingZoo">https://github.com/fhswf/MLPro-Int-PettingZoo</a>
<b>URL</b>	<a href="https://mlpro-int-pettingzoo.readthedocs.io">https://mlpro-int-pettingzoo.readthedocs.io</a>
<b>License</b>	Apache License 2.0

## MLPro-Int-River

MLPro: Integration River

<b>Topics</b>	machine-learning, mlpro-extension, online-machine-learning, river, wrapper
<b>Version</b>	v0.1.5 - Minor Fixes
<b>Last update</b>	Thu, 18 Apr 2024 18:15:40 GMT
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf/MLPro-Int-River">https://github.com/fhswf/MLPro-Int-River</a>
<b>URL</b>	<a href="https://mlpro-int-river.readthedocs.io">https://mlpro-int-river.readthedocs.io</a>
<b>License</b>	Apache License 2.0

## MLPro-Int-SB3

MLPro: Integration StableBaselines3

<b>Topics</b>	machine-learning, mlpro-extension, reinforcement-learning, stable-baselines3, wrapper
<b>Version</b>	v1.0.0 - Hello World!
<b>Last update</b>	Sun, 21 Apr 2024 15:09:59 GMT
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf/MLPro-Int-SB3">https://github.com/fhswf/MLPro-Int-SB3</a>
<b>URL</b>	<a href="https://mlpro-int-sb3.readthedocs.io">https://mlpro-int-sb3.readthedocs.io</a>
<b>License</b>	Apache License 2.0

## MLPro-Int-scikit-learn

MLPro: Integration scikit-learn

<b>Topics</b>	machine-learning, mlpro-extension, online-machine-learning, scikit-learn, wrapper
<b>Version</b>	v0.1.1 - Alignment with MLPro 1.4.0
<b>Last update</b>	Thu, 18 Apr 2024 19:40:12 GMT
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf/MLPro-Int-scikit-learn">https://github.com/fhswf/MLPro-Int-scikit-learn</a>
<b>URL</b>	<a href="http://mlpro-int-scikit-learn.readthedocs.io/">http://mlpro-int-scikit-learn.readthedocs.io/</a>
<b>License</b>	Apache License 2.0

## MLPro-MPPS

An extension feature of MLPro related to a simulation framework for multi-purpose production systems (MPPS)

<b>Topics</b>	machine-learning, mlpro-extension, python, simulation-environment
<b>Version</b>	v1.2.2
<b>Last update</b>	Wed, 21 Feb 2024 10:24:07 GMT
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf/MLPro-MPPS">https://github.com/fhswf/MLPro-MPPS</a>
<b>URL</b>	<a href="https://youtu.be/zjvDDBpl-bE">https://youtu.be/zjvDDBpl-bE</a>
<b>License</b>	Apache License 2.0

**mt\_baheti**

This repository is a part of master's thesis by Laxmikant Shrikant Baheti as a part of course Systems Engineering and Engineering Management

<b>Topics</b>	mlpro-extension
<b>Version</b>	v1.0.0 - Final Release for Master's Thesis
<b>Last update</b>	Sun, 29 Oct 2023 20:26:09 GMT
<b>URL (GitHub)</b>	<a href="https://github.com/fhswf/mt_baheti">https://github.com/fhswf/mt_baheti</a>
<b>URL</b>	
<b>License</b>	<a href="#">Apache License 2.0</a>

## 9.2 Single Contributors

This section lists MLPro extensions provided by single users.

## A1 - EXAMPLE POOL

## 10.1 MLPro-BF - Basic Functions

The following examples demonstrate various basic functionalities of MLPro-BF:

### 10.1.1 Layer 0 - Elementary Functions

## Various Functions

## Howto BF-001: Logging

## Executable code

```
## -----
↪ -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_001_logging.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2021-10-07  1.0.0     DA         Creation
## -- 2021-11-03  1.1.0     DA         Introduction of new log type C_LOG_TYPE_S for ↪
↪ success messages
## -- 2021-11-13  1.1.1     DA         Refactoring
## -- 2023-03-02  1.1.2     LSB        Refactoring
## -----
↪ -----

"""
Ver. 1.1.2 (2023-03-02)

This module demonstrates the Log class functionality.

You will learn:

1. How to use the log functionality of MLPro in custom implementations.
```

(continues on next page)

(continued from previous page)

```

2. How to use different log levels to log information, warnings and error.

3. How to switch logging functionality.
"""

from mlpro.bf.various import Log

# 1 Reuse logging property in your own class by inheriting from class Log
class MyClass(Log):

    # These constants are inherited from class Log and will be logged in every log line.
    ↪...
    C_TYPE = 'Demo class'
    C_NAME = 'MyClass'

    def __init__(self, p_logging=True):
        # The constructor of class Log initializes the internal logging and writes the ↪
        ↪first line...
        super().__init__(p_logging=p_logging)

    def my_method(self):
        # The log types I/E/W are also inherited from class Log...
        self.log(self.C_LOG_TYPE_I, 'Let me tell you what\'s going on...')
        self.log(self.C_LOG_TYPE_W, 'Something is weird...')
        self.log(self.C_LOG_TYPE_E, 'And here something failed...')
        self.log(self.C_LOG_TYPE_I, 'But don\'t worry. Everything is fine. It\'s just a ↪
        ↪demo:~')
        self.log(self.C_LOG_TYPE_S, 'This method terminated successfully!\n')

if __name__ == "__main__":

    # 2 Log everything inside your class...
    print('\n--\n-- Example 1: By default everything is logged...\n--\n')
    mc = MyClass(p_logging=Log.C_LOG_ALL)
    mc.my_method()

    # 3 Log nothing inside your class
    print('\n--\n-- Example 2: Now logging is switched off...\n--\n')
    mc.switch_logging(Log.C_LOG_NOTHING)
    mc.my_method()

    # 4 Log warnings and errors only

```

(continues on next page)

(continued from previous page)

```

print('\n--\n-- Example 3: Only warnings and errors are logged...\n--\n')
mc.switch_logging(Log.C_LOG_WE)
mc.my_method()

# 5 Log errors only
print('\n--\n-- Example 4: Only errors are logged...\n--\n')
mc.switch_logging(Log.C_LOG_E)
mc.my_method()

# 6 Log everything again
print('\n--\n-- Example 5: Everything is logged again...\n--\n')
mc.switch_logging(Log.C_LOG_ALL)
mc.my_method()

```

## Results

```

2022-09-02 14:19:26.962434 I Demo class MyClass: Let me tell you what's going on...
2022-09-02 14:19:26.962437 W Demo class MyClass: Something is weird...
2022-09-02 14:19:26.962500 E Demo class MyClass: And here something failed...
2022-09-02 14:19:26.962505 I Demo class MyClass: But don't worry. Everything is fine. It's just a demo:)
2022-09-02 14:19:26.962508 S Demo class MyClass: This method terminated successfully!

```

## Cross Reference

- *API Reference: Various*

## Howto BF-002: Timer

### Executable code

```

## -----
↪-----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_002_timer.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2021-05-19  1.0.0     DA         Creation
## -- 2021-09-11  1.0.0     MRD        Change Header information to match our new library.↪
↪name
## -- 2021-11-13  1.0.1     DA         Minor fix
## -- 2023-03-02  1.0.2     LSB        Refactoring
## -----
↪-----

"""
Ver. 1.0.2 (2023-03-02)

This module demonstrates the Timer class functionality.

You will learn:

```

(continues on next page)

(continued from previous page)

```

1. How to use the timer functionality of MLPro in native and custom implementations.
2. How to use the lap functionality of MLPro's Timer.
"""

import time
import random
from datetime import timedelta
from mlpro.bf.various import Timer, Log

# Demo class
class TimerDemo (Log):

    C_TYPE = 'Demo class'
    C_NAME = 'Timer'

    def __init__(self, p_timer:Timer):
        self.timer = p_timer
        super().__init__()

    def log(self, p_type, *p_args):
        super().log(p_type, 'Process time', self.timer.get_time(), ', Cycle', self.timer.
↪get_lap_id(), 'Lap time', self.timer.get_lap_time(), '--', *p_args)

    def run_step(self, p_step_id):
        self.log(self.C_LOG_TYPE_I, 'Process step', p_step_id, 'started')
        duration = 0.6 * random.random()
        time.sleep(duration)
        self.log(self.C_LOG_TYPE_I, 'Process step', p_step_id, 'ended after', duration,
↪'seconds')

    def run_cycle(self):
        self.run_step(1)
        self.run_step(2)
        self.run_step(3)
        if not self.timer.finish_lap():
            self.log(self.C_LOG_TYPE_W, 'Last process cycle timed out!!')

    def run(self):
        for i in range(10):
            self.run_cycle()

```

(continues on next page)



(continued from previous page)

```

if __name__ == "__main__":

    # Example 1
    print('\n\nExample 1: Timer in virtual time mode with lap duration 1 day and 15_
↪seconds...\n\n')
    t = Timer(Timer.C_MODE_VIRTUAL, timedelta(1,15,0))
    d = TimerDemo(t)
    d.run()

    # Example 2
    print('\n\nExample 2: Timer in real time mode with lap duration 1 second and_
↪forced timeout situations...\n\n')
    t = Timer(Timer.C_MODE_REAL, timedelta(0,1,0))
    d = TimerDemo(t)
    d.run()

```

## Results

```

Example 1: Timer in virtual time mode with lap duration 1 day and 15 seconds...

2023-02-10 15:10:52.396617 I Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Instantiated
2023-02-10 15:10:52.396617 I Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 1 started
2023-02-10 15:10:52.981296 I Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 1 ended after 0.5727354329679715 seconds
2023-02-10 15:10:52.981296 I Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 2 started
2023-02-10 15:10:53.250889 I Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 2 ended after 0.2671762029006218 seconds
2023-02-10 15:10:53.250889 I Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 3 started
2023-02-10 15:10:53.723793 I Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 3 ended after 0.46618259516486654 seconds
2023-02-10 15:10:53.723793 I Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 1 started
2023-02-10 15:10:54.310007 I Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 1 ended after 0.5742077882721884 seconds
2023-02-10 15:10:54.310007 I Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 2 started
2023-02-10 15:10:54.533070 I Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 2 ended after 0.21479102210586493 seconds
2023-02-10 15:10:54.533070 I Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 3 started
2023-02-10 15:10:54.612608 I Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 3 ended after 0.06589391986663488 seconds
2023-02-10 15:10:54.612608 I Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 1 started
2023-02-10 15:10:54.961244 I Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 1 ended after 0.33888097514695076 seconds
2023-02-10 15:10:54.961244 I Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 2 started
2023-02-10 15:10:55.278839 I Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 2 ended after 0.3068213153874194 seconds
2023-02-10 15:10:55.278839 I Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 3 started
2023-02-10 15:10:55.706568 I Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 3 ended after 0.4120912986840059 seconds

```

## Cross Reference

- *API Reference: Various*

## Howto BF-003: Store and plot data

### Prerequisites

PLease install following packages to run this howto

- `Matplotlib`

### Executable code

```

## -----
↪-----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪learning

```

(continues on next page)

(continued from previous page)

```

## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_003_store_plot_and_save_variables.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      DeScription
## -- 2021-06-16  1.0.0     SY         Creation/Release
## -- 2021-06-21  1.1.0     SY         Adjustment to updated DataPlotting class
## -- 2021-07-01  1.2.0     SY         Adjustment due to extension in save and load data
## -- 2021-09-11  1.2.1     MRD        Change Header information to match our new library.↪
↪ name
## -- 2021-10-25  1.2.2     SY         Adjustment due to improvement in DataPlotting
## -- 2021-10-26  1.2.3     SY         Rename module
## -- 2023-03-02  1.2.4     LSB        Refactoring
## -----
↪ -----

"""
Ver. 1.2.4 (2023-03-02)

This module demonstrates how to store, plot, save and load variables.

You will learn:

1. How to use the DataStoring class of MLPro and its functionalities.
2. How to add frames and data names in the data storing object.
3. How to memorize data using the DataStoring class.
4. How to plot the memorized frame from DataStoring object.
"""

from mlpro.bf.various import *
from mlpro.bf.data import *
from mlpro.bf.plot import *
import random

if __name__ == "__main__":
    num_eps      = 10
    num_cycles    = 10000
    data_names    = ["reward", "states_1", "states_2", "model_loss"]
    data_printing = {"reward":      [True, 0, 10],
                    "states_1":    [True, 0, 4],
                    "states_2":    [True, 0, 4],
                    "model_loss":  [True, 0, -1]}

    ## 1. How to store data ##
    mem = DataStoring(data_names)
    for ep in range(num_eps):

```

(continues on next page)

(continued from previous page)

```

ep_id = ("ep. %s"%str(ep+1))
mem.add_frame(ep_id)
for i in range(num_cycles):
    mem.memorize("reward",ep_id,random.uniform(0+(ep*0.5),5+(ep*0.5)))
    mem.memorize("states_1",ep_id,random.uniform(2-(ep*0.2),4-(ep*0.2)))
    mem.memorize("states_2",ep_id,random.uniform(0+(ep*0.2),2+(ep*0.2)))
    mem.memorize("model_loss",ep_id,random.uniform(0.25-(ep*0.02),1-(ep*0.07)))

## 2. How to plot stored data ##
# 2.1. Plotting data per cycle
# mem_plot = DataPlotting(mem, p_type=DataPlotting.C_PLOT_TYPE_CY, p_window=100,
#
#                               p_showing=True, p_printing=data_printing, p_figsize=(7,
↪7),
#
#                               p_color="darkblue")
# 2.2. Plotting data with continuous cycle
mem_plot = DataPlotting(mem, p_type=DataPlotting.C_PLOT_TYPE_EP, p_window=1000,
↪p_showing=True, p_printing=data_printing, p_figsize=(7,
7),
#
#                               p_color="darkblue")
# 2.3. Plotting data per episode according to its mean value
# mem_plot = DataPlotting(mem, p_type=DataPlotting.C_PLOT_TYPE_EP_M, p_window=1,
#
#                               p_showing=True, p_printing=data_printing, p_figsize=(7,
↪7),
#
#                               p_color="darkblue")
mem_plot.get_plots()

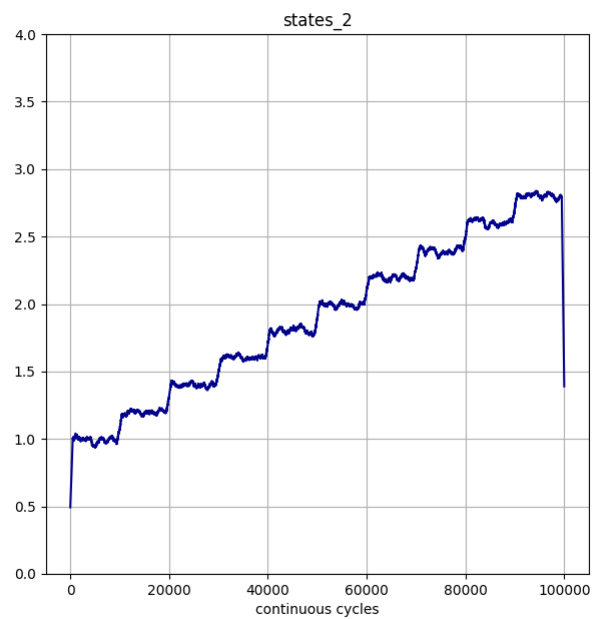
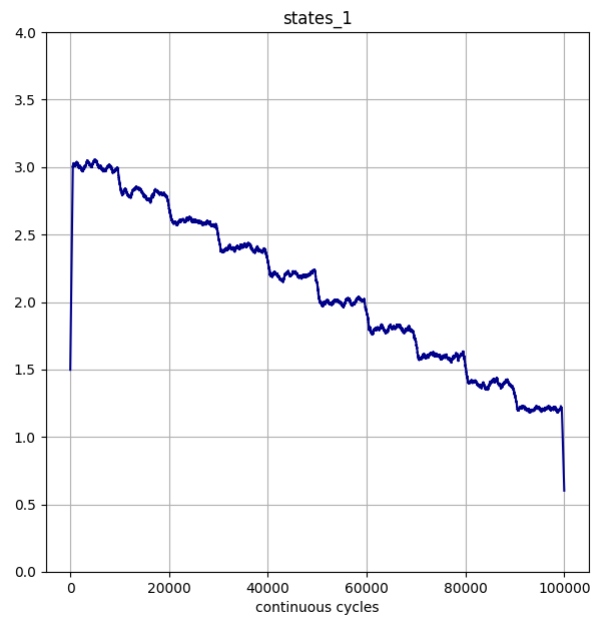
## 3. How to save plots and data in binary file (variables, classes, etc.) ##
path_save = input("Input path_save : ")
#Do not include quote-unquote ("" or " ) into target path name
mem_plot.save_plots(path_save, "pdf")
mem_plot.save(path_save, "plot_memory")
mem.save(path_save, "data_memory")
mem.save_data(path_save, "data_storage", "\t")

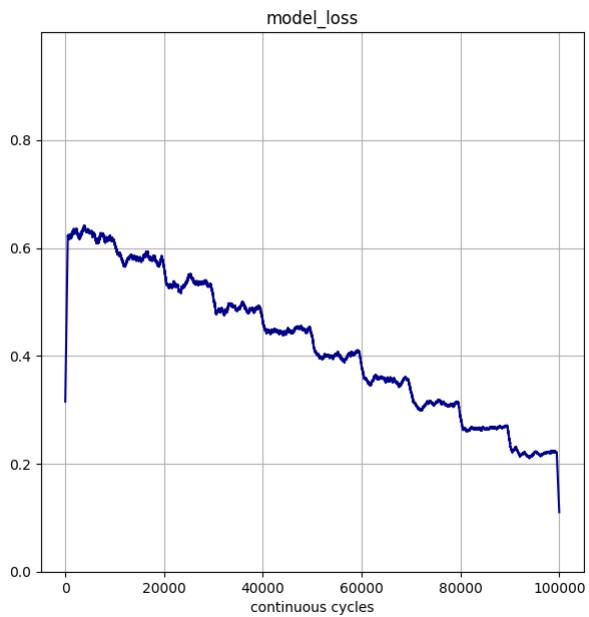
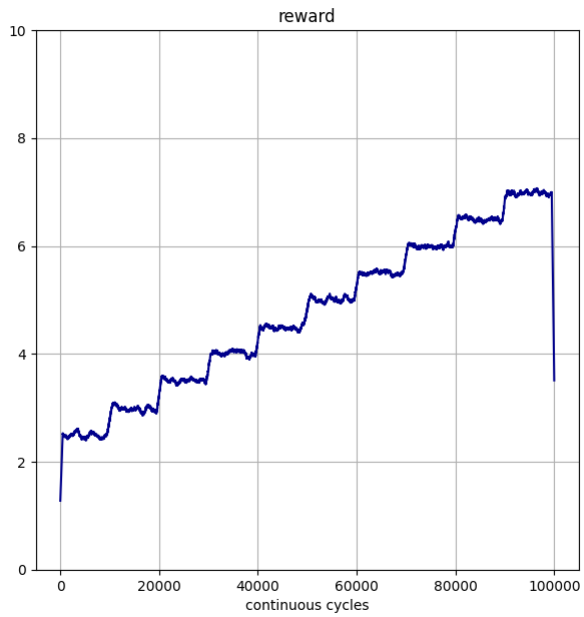
## 4. How to load data from binary file ##
path_load = path_save
mem_load = DataStoring.load(path_load, "data_memory")
print("Comparison :")
print("Original data          : %.5f"%mem.memory_dict["reward"]["ep. 1"][0])
print("Loaded data from binary file : %.5f"%mem_load.memory_dict["reward"]["ep. 1
↪"] [0])

## 5. How to load data from csv file ##
data_names = []
mem_from_csv = DataStoring(data_names)
mem_from_csv.load_data(path_load, "data_storage.csv", "\t")
print("Comparison :")
print("Original data          : %.5f"%mem.memory_dict["reward"]["ep. 1"][0])
print("Loaded data from csv file : %.5f"%mem_from_csv.memory_dict["reward"]["ep. 1
↪"] [0])

```

## Results





## Cross Reference

- *API Reference: Various*

## Howto BF-004: Buffers

### Executable code

```
## -----
↪-----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪learning
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_004_buffers.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2021-10-26  1.0.0     SY         Creation/Release
## -- 2023-03-02  1.0.1     LSB        Refactoring
## -----
↪-----

"""
Ver. 1.0.1 (2023-03-02)

This module demonstrates how to use classes Buffer and BufferElement.

You will learn:

1. How to use the buffer, buffer element class of MLPro in native and custom_
↪implementations.

2. How to add values to a buffer element object.

2. How to add buffer elements with data to the buffer object.

3. How to get the buffer elements and data from a buffer object and clear a buffer_
↪object.
"""

from mlpro.bf.data import *
import random

if __name__ == "__main__":

    # 1. Instantiate a buffer with random sampling
    buffer      = BufferRnd(p_size=100)
    num_cycles = 150

    # 2 Generate random values and store them to the Buffer
    for i in range(num_cycles):
```

(continues on next page)

(continued from previous page)

```

# 2.1 Store the values and their names in a BufferElement
buffer_element = BufferElement({"reward":random.uniform(-10,10),
                                "actions":[random.uniform(0,1),random.uniform(0,
→1)]]})

# 2.2 Example: add value element in the developed BufferElement
buffer_element.add_value_element(dict(accuracy=random.uniform(0,1)))

# 2.3 Add the BufferElement into the Buffer
buffer.add_element(buffer_element)
print('Cycle : %.i'%int(i+1))

# 2.4 Checking whether buffer is full or not
if not buffer.is_full():
    print('Buffer is not full yet, keep collecting data!\n')
else:
    print('Buffer is full, ready to use!')

# 2.5 Get all data from the Buffer
all_data = buffer.get_all()
_actions      = all_data["actions"]
_reward       = all_data["reward"]
_accuracy     = all_data["accuracy"]

# 2.6 Get sample data from the Buffer, you define your sampling strategy by
# redefining method _gen_sample_ind(self, p_num:int)
sample_data = buffer.get_sample(p_num=10)
print('Get sample!\n')
_actions_sample = sample_data["actions"]
_reward_sample  = sample_data["reward"]
_accuracy_sample = sample_data["accuracy"]

# 3 To clear your buffer
if buffer is not None:
    buffer.clear()
    print('Buffer is cleared!')

```

## Results

```
Cycle : 96
Buffer is not full yet, keep collecting data!

Cycle : 97
Buffer is not full yet, keep collecting data!

Cycle : 98
Buffer is not full yet, keep collecting data!

Cycle : 99
Buffer is not full yet, keep collecting data!

Cycle : 100
Buffer is full, ready to use!
Get sample!

Cycle : 101
Buffer is full, ready to use!
Get sample!

Cycle : 102
Buffer is full, ready to use!
Get sample!
```

### Cross Reference

- *API Reference: Various*

### Howto BF-005: Persistence

#### Executable code

```
## -----
↪ -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_005_persistence.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2023-03-22  1.0.0     DA         Creation/Release
## -----
↪ -----

"""
Ver. 1.0.0 (2023-03-22)
```

(continues on next page)



(continued from previous page)

*This module demonstrates the basic persistence functionalities of MLPro.*

*You will learn:*

- 1. How to create an own persistent custom class*
- 2. How to save an object of your custom class to a file*
- 3. How to load an object of your custom class from a file*
- 4. How to implement custom methods to separately save internal data that can not be ↪ pickled*

"""

```
from mlpro.bf.various import *
from datetime import datetime
from pathlib import Path
```

```
## -----
↪ -----
```

```
## -----
↪ -----
```

```
class MyClass (Persistent):
```

```
    """
```

```
    Own custom class that uses MLPro's persistence...
```

```
    """
```

```
    C_TYPE      = 'Custom'
    C_NAME      = 'Myclass'
```

```
## -----
↪ -----
```

```
    def __init__(self, p_id=None, p_logging=Log.C_LOG_ALL):
        super().__init__(p_id, p_logging)
        self._data = {}
        self._separate_data = {}
```

```
## -----
↪ -----
```

```
    def set_data(self, **p_kwargs):
        self._data = p_kwargs.copy()
```

```
## -----
↪ -----
```

```
    def set_separate_data(self, **p_kwargs):
```

(continues on next page)

(continued from previous page)

```

        self._separate_data = p_kwargs.copy()

## -----
↪ -----
    def log_data(self):
        self.log(Log.C_LOG_TYPE_I, 'Data to be pickled:\n\n', self._data, '\n')
        self.log(Log.C_LOG_TYPE_I, 'Separate data:\n\n', self._separate_data, '\n')

## -----
↪ -----
    def _complete_state(self, p_path: str, p_os_sep: str, p_filename_stub: str):

        # Complete object state from separate external data file
        self._separate_data = {}

        with open(p_path + p_os_sep + p_filename_stub + '.dat', 'r') as f:
            for line in f:
                (key, value) = line.split(sep='=')
                (value, nl) = value.split(sep='\n')
                self._separate_data[key] = value

## -----
↪ -----
    def _reduce_state(self, p_state: dict, p_path: str, p_os_sep: str, p_filename_stub: str):

        # 1 Persist all separate data that can/shall not be pickled
        with open(p_path + p_os_sep + p_filename_stub + '.dat', 'w') as f:
            for key in p_state['_separate_data'].keys():
                f.write(key + '=' + p_state['_separate_data'][key] + '\n')

        # 2 Remove separate data from object state
        del p_state['_separate_data']

## -----
↪ -----
    def __del__(self):
        try:
            self.log(Log.C_LOG_TYPE_W, 'I just died in our arms tonight...')
        except:
            pass

# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode

```

(continues on next page)

(continued from previous page)

```

logging = Log.C_LOG_ALL

else:
    # 1.2 Parameters for internal unit test
    logging = Log.C_LOG_NOTHING

now      = datetime.now()
path     = str(Path.home()) + os.sep + '%04d-%02d-%02d %02d.%02d.%02d ' % (now.year, now.
    month, now.day, now.hour, now.minute, now.second) + ' MLPro Persistence Test'

# 2 Instantiate the demo object
mc = MyClass(p_logging=logging)

# 2.1 Persistent classes in MLPro have unique id and filename...
mc.log(Log.C_LOG_TYPE_I, 'My unique Id:', str(mc.get_id()))
mc.log(Log.C_LOG_TYPE_I, 'My unique filename:', mc.get_filename())

# 3 Store data to demo object
mc.set_data( p1='Hello', p2='World!' )
mc.set_separate_data( p1='How', p2='are', p3='you', p4='today?' )
mc.log_data()

# 4 Save demo object to file
mc.save(p_path=path)
filename = mc.get_filename()
mc = None

# 5 Reload same demo object from file
mc = MyClass.load(p_path=path, p_filename=filename)
mc.log(Log.C_LOG_TYPE_I, 'My unique Id:', str(mc.get_id()))
mc.log(Log.C_LOG_TYPE_I, 'My unique filename:', mc.get_filename())
mc.log_data()

```

## Results

```

2023-03-24 17:21:25.881506 I Custom "Myclass": Instantiated
2023-03-24 17:21:25.881529 I Custom "Myclass": My unique Id: 58f78c69-3577-4097-b799-8229e8e12e26
2023-03-24 17:21:25.881538 I Custom "Myclass": My unique filename: MyClass[58f78c69-3577-4097-b799-8229e8e12e26].pk1
2023-03-24 17:21:25.881545 I Custom "Myclass": Data to be pickled:
    {'p1': 'Hello', 'p2': 'World!'}

2023-03-24 17:21:25.881556 I Custom "Myclass": Separate data:
    {'p1': 'How', 'p2': 'are', 'p3': 'you', 'p4': 'today?'}

2023-03-24 17:21:25.882966 I Custom "Myclass": Object saved to file "/home/detlef/2023-03-24 17.21.25 MLPro Persistence Test/MyClass[58f78c69-3577-4097-b799-8229e8e12e26].pk1"
2023-03-24 17:21:25.882976 W Custom "Myclass": I just died in our arms tonight...
2023-03-24 17:21:25.883127 I Custom "Myclass": Object loaded from file "/home/detlef/2023-03-24 17.21.25 MLPro Persistence Test/MyClass[58f78c69-3577-4097-b799-8229e8e12e26].pk1"
2023-03-24 17:21:25.883139 I Custom "Myclass": My unique Id: 58f78c69-3577-4097-b799-8229e8e12e26
2023-03-24 17:21:25.883146 I Custom "Myclass": My unique filename: MyClass[58f78c69-3577-4097-b799-8229e8e12e26].pk1
2023-03-24 17:21:25.883151 I Custom "Myclass": Data to be pickled:
    {'p1': 'Hello', 'p2': 'World!'}

2023-03-24 17:21:25.883159 I Custom "Myclass": Separate data:
    {'p1': 'How', 'p2': 'are', 'p3': 'you', 'p4': 'today?'}

```

## Cross Reference

- *API Reference: Various*

## User Interaction

### Howto BF-UI-001: SciUI - Reuse of interactive 2D/3D Input Space

#### Prerequisites

Please install the following packages to run this example properly:

- Matplotlib
- Tkinter

#### Executable code

```
## -----
↪-----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_ui_001_reuse_of_interactive_2d_3d_input_space.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2021-06-20  0.0.0     DA         Creation
## -- 2021-07-03  1.0.0     DA         Release of first version
## -- 2021-09-11  1.0.0     MRD        Change Header information to match our new library.
↪name
## -- 2022-01-06  1.0.1     DA         Corrections
## -- 2022-03-21  1.0.2     SY         Refactoring following class Dimensions update
## -- 2022-10-08  1.0.3     DA         Refactoring following class Dimensions update
## -----
↪-----

"""
Ver. 1.0.3 (2022-10-08)

Demo scenarios for SciUI framework that shows the reuse of the interactive 2D/3D input
↪space class.
Can be executed directly...
"""

from mlpro.bf.ui.sciui.framework import *
from mlpro.bf.ui.sciui.pool.iis import InteractiveInputSpace
from mlpro.bf.math import *

## -----
↪-----
```

(continues on next page)

(continued from previous page)

```

## -----
↪ -----
class DemoIIS2D(SciUIScenario):

    C_NAME          = 'Demo for interactive 2D Input Space'
    C_VERSION        = '1.0.0'
    C_RELEASED       = True
    C_VISIBLE        = True

## -----
↪ -----
    def init_component(self):
        super().init_component()

        # 1 Add scenario-specific variables to shared db
        InteractiveInputSpace.enrich_shared_db(self.shared_db)
        self.shared_db.iis_isspace.add_dim( Dimension( p_name_short='x1',
                                                       p_description='',
                                                       p_name_latex='x_1',
                                                       p_unit='m',
                                                       p_unit_latex='m',
                                                       p_boundaries=[-5,5]) )

        self.shared_db.iis_isspace.add_dim( Dimension( p_name_short='x2',
                                                       p_description='',
                                                       p_name_latex='x_2',
                                                       p_unit='m/s',
                                                       p_unit_latex='\\frac{m}{s}',
                                                       p_boundaries=[-25,25]) )

        # 2 Build scenario structure
        self.add_component(InteractiveInputSpace(self.shared_db, p_row=0, p_col=0, p_
↪ padx=5, p_logging=self._level))

## -----
↪ -----
## -----
↪ -----
class DemoIIS3D(SciUIScenario):

    C_NAME          = 'Demo for interactive 3D Input Space'
    C_VERSION        = '1.0.0'
    C_RELEASED       = True
    C_VISIBLE        = True

## -----
↪ -----
    def init_component(self):
        super().init_component()

```

(continues on next page)

(continued from previous page)

```

# 1 Add scenario-specific variables to shared db
InteractiveInputSpace.enrich_shared_db(self.shared_db)
self.shared_db.iis_ispace.add_dim( Dimension( p_name_short='x1',
                                              p_description='',
                                              p_name_latex='x_1',
                                              p_unit='m',
                                              p_unit_latex='m',
                                              p_boundaries=[-5,5]) )

self.shared_db.iis_ispace.add_dim( Dimension( p_name_short='x2',
                                              p_description='',
                                              p_name_latex='x_2',
                                              p_unit='m/s',
                                              p_unit_latex='\\frac{m}{s}',
                                              p_boundaries=[-25,25]) )

self.shared_db.iis_ispace.add_dim( Dimension( p_name_short='x3',
                                              p_description='',
                                              p_name_latex='x_3',
                                              p_unit='m/s^2',
                                              p_unit_latex='\\frac{m}{s^2}',
                                              p_boundaries=[-15,15]) )

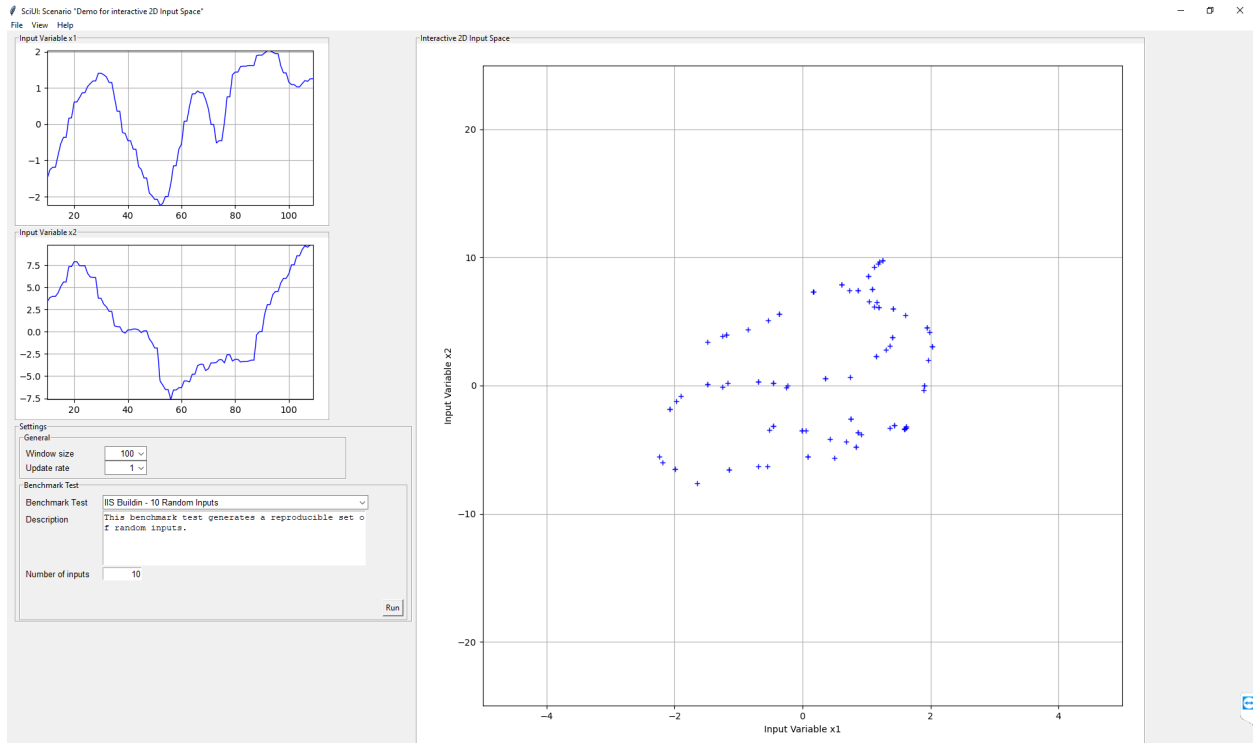
# 2 Build scenario structure
self.add_component(InteractiveInputSpace(self.shared_db, p_row=0, p_col=0, p_
↪padx=5, p_logging=self._level))

if (__name__ == '__main__'):
    from mlpro.bf.ui.sciui.main import SciUI
    SciUI()

```

## Results

The SciUI application should start and show an interactive demo of a 2D/3D input space as follows:



## Cross Reference

- [API Reference: SciUi](#)

## 10.1.2 Layer 1 - Computation

### Event Handling

#### Howto BF-EH-001: Event Handling

### Executable code

#### Results

```
2023-02-10 15:50:45.050902 I Event handler "My handler": Instantiated
2023-02-10 15:50:45.050902 I EventManager "My main class": Instantiated
2023-02-10 15:50:45.050902 I EventManager "My main class": Event "MYEVENT" fired
2023-02-10 15:50:45.050902 I EventManager "My main class": Calling handler 0
2023-02-10 15:50:45.050902 I Event handler "My handler": Received event id MYEVENT
2023-02-10 15:50:45.051902 I Event handler "My handler": Event data: {'p_par1': 'Hello', 'p_par2': 'World!'}
2023-02-10 15:50:45.051902 I EventManager "My main class": Event "MYEVENT" fired
2023-02-10 15:50:45.051902 I EventManager "My main class": No handlers registered for event "MYEVENT"
```

## Cross Reference

- [API Reference: Event Handling](#)

## Multitasking

### Howto BF-MT-001: Multitasking - Parallel Algorithms

#### Prerequisites

Please install following packages to run this howto

- Multiprocess

#### Executable code

```
## -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_mt_001_parallel_algorithms.py
## -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-03  1.0.0     DA       Creation/release
## -- 2022-10-09  1.1.0     DA       Simplification
## -- 2022-10-12  1.2.0     DA       Restructuring of demo steps
## -- 2022-10-13  1.3.0     DA       Restructuring of demo steps
## -----

"""
Ver. 1.3.0 (2022-10-13)

This module demonstrates the use of classes ASync and Shared as part of MLPro's
↳multitasking concept.
Both classes are used to implement a simple parallel algorithm class MyParallelAlgorithm
↳with a
method _async_subtask() for asynchronous execution collecting results in a shared object.

In three runs method _async_subtask() is executed several times a) synchronously, b) as
↳threads and
c) as processes. Depending on the number of cores per cpu, the operating system, and
↳further factors
multiprocessing outperforms multithreading more ore less drastically. Method
↳MyParallelAlgorithm.execute()
determines and logs the speed factor of multithreading and multiprocessing in comparison
↳to serial/synchronous
computation. Open the perfmeter of your system and play with number of tasks and their
↳duration to observe
the behavior.

All sub-tasks store dummy results in a shared object. It is no surprise that the order
↳of result entries
in multithreading and multiprocessing mode does not 100% match the order of sub-task
↳starts.
```

(continues on next page)



(continued from previous page)

You will learn:

- 1) The meaning and basic properties of the classes Async and Shared.
- 2) How to set up an own class with parallel running sub-tasks inside.
- 3) How to collect results of the parallel sub-functions in a shared object.

```

"""

from time import sleep
from mlpro.bf.various import Log
import multiprocessing as mp
import mlpro.bf.mt as mt
from datetime import datetime, timedelta
from cmath import pi, sin, cos, tan
import random

## -----
↪ -----
## -----
↪ -----

class MyParallelAlgorithm (mt.Async):
    """
    This class demonstrates how to run methods asynchronously and to collect results in_
    ↪ a shared
    object.
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_TYPE      = 'Demo'
    C_NAME      = 'Parallel Algorithm'

## -----
↪ -----

    def __init__( self,
                  p_num_tasks:int,
                  p_duration:timedelta,
                  p_range_max=mt.Async.C_RANGE_PROCESS,
                  p_class_shared=mt.Shared,
                  p_logging=Log.C_LOG_ALL ):

        super().__init__( p_range_max=p_range_max,
                          p_class_shared=p_class_shared,
                          p_logging=p_logging )

        self._num_tasks      = p_num_tasks
        self._duration        = p_duration
        self._duration_sec    = self._duration.seconds + self._duration.microseconds /_

```

(continues on next page)

(continued from previous page)

```

↪10000000

## -----
↪-----
    def execute(self):
        # Log at the beginning of a run
        if self._range == self.C_RANGE_NONE:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'synchronous_
↪tasks started')
        elif self._range == self.C_RANGE_THREAD:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'asynchronous_
↪tasks as threads started')
        else:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'asynchronous_
↪tasks as processes started')

        # Start number of tasks (a)synchronously
        time_start = datetime.now()
        for t in range(self._num_tasks):
            self._start_async( p_target=self._async_subtask, p_tid=t)

        self.wait_async_tasks()
        time_end = datetime.now()

        # Determination of speed factor (no parallelism = 1)
        duration_real = time_end - time_start
        duration_real_sec = duration_real.seconds + duration_real.microseconds /
↪1000000
        speed_factor = round( self._num_tasks * self._duration_sec / duration_
↪real_sec, 2)

        # Log of speed factor
        if self._range == self.C_RANGE_NONE:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'synchronous_
↪tasks ended (speed factor =', speed_factor, ')')
        elif self._range == self.C_RANGE_THREAD:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'asynchronous_
↪tasks as threads ended (speed factor =', speed_factor, ')')
        else:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'asynchronous_
↪tasks as processes ended (speed factor =', speed_factor, ')')

        # Log of results collected in the shared object
        self.log(Log.C_LOG_TYPE_I, 'Results in shared object are:\n',self._so.get_
↪results())

## -----
↪-----
    def _async_subtask(self, p_tid):

```

(continues on next page)

(continued from previous page)

```

self.log(Log.C_LOG_TYPE_I, 'Task', p_tid, 'started')

# 1 Sub-task needs to check in on shared object
self._so.checkin( p_tid=p_tid )

# 2 Dummy implementation to simulate a busy sub-task
time_start = datetime.now()
result      = 0

while True:
    # do something meaningful
    for i in range(300):
        result += sin(random.random()*pi) * cos(random.random()*pi) * tan(random.
↪random()*pi)

        time_current = datetime.now()
        time_diff     = time_current - time_start
        if time_diff >= self._duration: break

# 3 Sub-task can optionally store results in the shared object.
self._so.add_result(p_tid=p_tid, p_result=result)

# 4 Sub-task needs to check out from shared object
self._so.checkout( p_tid=p_tid )

self.log(Log.C_LOG_TYPE_I, 'Task', p_tid, 'stopped')

if __name__ == "__main__":

    # 1 Preparation of execution

    # https://docs.python.org/3/library/multiprocessing.html?highlight=freeze\_support
    ↪#multiprocessing.freeze_support
    mp.freeze_support()

    num_tasks    = 20
    duration      = timedelta(0,1,0)
    pause_sec     = 5
    logging       = Log.C_LOG_ALL

    # 2 Execution of demo class (synchronously)
    a = MyParallelAlgorithm( p_num_tasks = num_tasks,
                             p_duration = duration,
                             p_range_max = mt.Async.C_RANGE_NONE,
                             p_logging  = logging )

```

(continues on next page)

(continued from previous page)

```

a.execute()

# 3 Execution of demo class (asynchronously, multithreading)
a.log(Log.C_LOG_TYPE_W, 'Short break for better observation of CPU load in perfmeter
↪')
sleep(pause_sec)

a = MyParallelAlgorithm( p_num_tasks = num_tasks,
                        p_duration = duration,
                        p_range_max = mt.Async.C_RANGE_THREAD,
                        p_logging = logging )

a.execute()

# 4 Execution of demo class (asynchronously, multiprocessing)
a.log(Log.C_LOG_TYPE_W, 'Short break for better observation of CPU load in perfmeter
↪')
sleep(pause_sec)

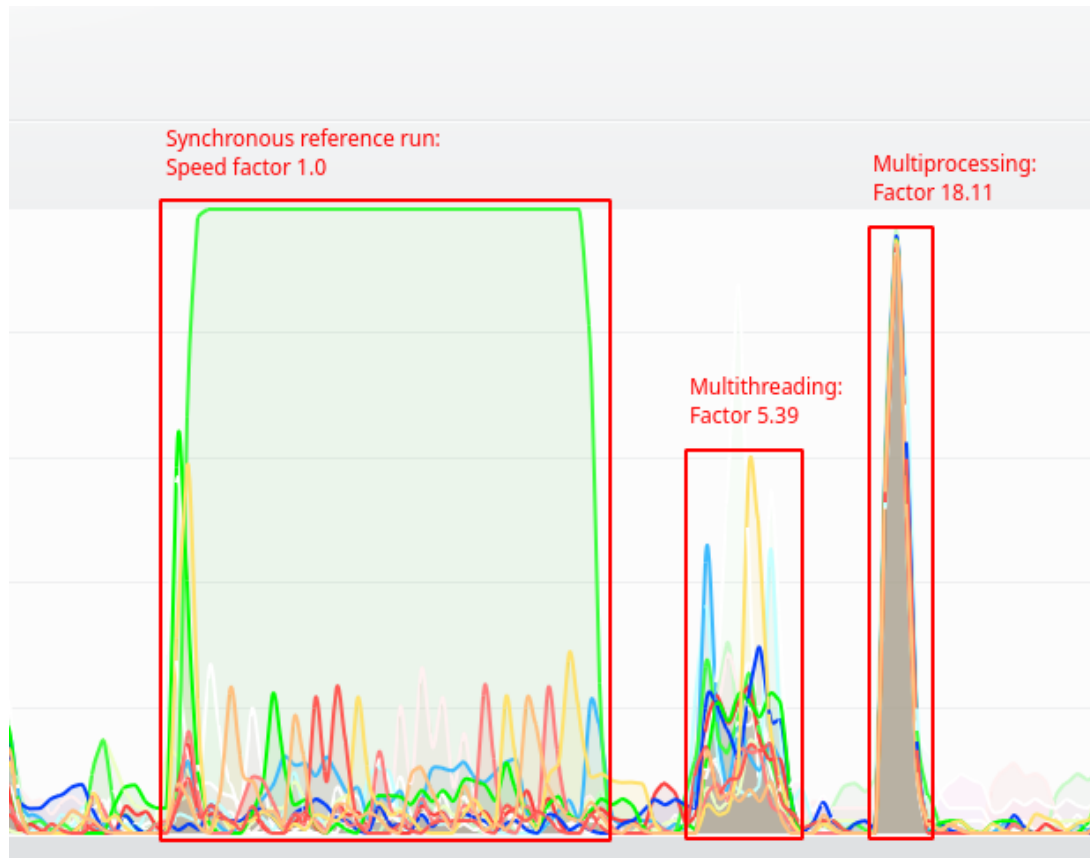
a = MyParallelAlgorithm( p_num_tasks = num_tasks,
                        p_duration = duration,
                        p_range_max = mt.Async.C_RANGE_PROCESS,
                        p_logging = logging )

a.execute()

```

## Results

The howto example logs details of the three runs and in particular the speed factors of multithreading and multiprocessing in comparison to the serial/synchronous execution. On a PC with an AMD Ryzen 7 CPU (8/16 cores) running Linux, the system monitor shows an approx. 5x speedup with multithreading and an approx. 18x speedup with multiprocessing.



### Cross Reference

- *API Reference: Multitasking*

## Howto BF-MT-002: Multitasking - Tasks and Workflows

### Prerequisites

To run this howto please install the following packages

- `Multiprocess`

### Executable code

```
## -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_mt_002_tasks_and_workflows.py
## -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-04  1.0.0     DA       Creation/release
## -- 2022-10-09  1.1.0     DA       Simplification
## -- 2022-10-12  1.2.0     DA       Restructuring of demo steps
## -- 2022-10-13  1.3.0     DA       Simplification and reduction to multithreading
```

(continues on next page)

(continued from previous page)

```

## -- 2022-11-07  1.3.1    DA      Minor correction
## -----
↪-----

"""
Ver. 1.3.1 (2022-11-07)

This module demonstrates the use of tasks and workflows as part of MLPro's multitasking_
↪concept.
To this regard, a demo custom task class is implemented. At first the task class is_
↪instantiated 9
times, added to a workflow, and chained by predecessor relations. In two experiments the_
↪workflow is
executed synchronously and in multithreading mode. In the latter case, the tasks are_
↪partly executed
parallel which increases the computation performance.

In both experiments pseudo results are stored in a shared object.

You will learn:

1) How to implement an own custom task
2) How to store results in a shared object
3) How to add tasks to a workflow
4) How to run tasks and workflows in various ranges of asynchronicity

"""

from time import sleep
from mlpro.bf.various import Log
import mlpro.bf.mt as mt
from datetime import datetime, timedelta
from cmath import pi, sin, cos, tan
import random

## -----
↪-----
## -----
↪-----

class MyTask (mt.Task):
    """
    Demo implementation of a task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)

```

(continues on next page)

(continued from previous page)

```

C_NAME      = 'My task'

## -----
↪ -----
def __init__( self,
               p_duration:timedelta,
               p_name:str=None,
               p_range_max: int = mt.Task.C_RANGE_THREAD,
               p_autorun=mt.Task.C_AUTORUN_NONE,
               p_class_shared=None,
               p_visualize:bool=False,
               p_logging=Log.C_LOG_ALL ):

    super().__init__( p_name=p_name,
                      p_range_max=p_range_max,
                      p_autorun=p_autorun,
                      p_class_shared=p_class_shared,
                      p_visualize=p_visualize,
                      p_logging=p_logging )

    self._duration = p_duration

## -----
↪ -----
def _run(self, **p_kwargs):

    tid = self.get_tid()

    # 1 Dummy implementation to simulate a busy sub-task
    time_start = datetime.now()
    result      = 0

    while True:
        # do something meaningful
        for i in range(500):
            result += sin(random.random()*pi) * cos(random.random()*pi) * tan(random.
↪random()*pi)

            time_current = datetime.now()
            time_diff    = time_current - time_start
            if time_diff >= self._duration: break

    # 3 Sub-task can optionally store results in the shared object.
    self._so.add_result(p_tid=self.get_name(), p_result=result)

# 1 Preparation of execution
if __name__ == '__main__':

```

(continues on next page)

(continued from previous page)

```

# 1.1 Preparation for demo mode
duration    = timedelta(0,1,0)
pause_sec   = 5
logging     = Log.C_LOG_ALL

else:
# 1.2 Preparation for unit test mode
num_tasks   = 2
duration    = timedelta(0,0,10000)
pause_sec   = 0
logging     = Log.C_LOG_NOTHING

# 2 Creation of a workflow with 9 tasks within

# 2.1 Creation of 9 tasks
t1a = MyTask( p_duration=duration, p_name='t1a', p_logging=logging )
t1b = MyTask( p_duration=duration, p_name='t1b', p_logging=logging )
t1c = MyTask( p_duration=duration, p_name='t1c', p_logging=logging )

t2a = MyTask( p_duration=duration, p_name='t2a', p_logging=logging )
t2b = MyTask( p_duration=duration, p_name='t2b', p_logging=logging )
t2c = MyTask( p_duration=duration, p_name='t2c', p_logging=logging )

t3a = MyTask( p_duration=duration, p_name='t3a', p_logging=logging )
t3b = MyTask( p_duration=duration, p_name='t3b', p_logging=logging )
t3c = MyTask( p_duration=duration, p_name='t3c', p_logging=logging )

# 2.2 Create a workflow and add the tasks
wf = mt.Workflow( p_name='wf1',
                  p_range_max=mt.Workflow.C_RANGE_THREAD,
                  p_class_shared=mt.Shared,
                  p_logging=logging )

# 2.2.1 At first we add three tasks that build the starting points of our workflow
wf.add_task( p_task=t1a )
wf.add_task( p_task=t1b )
wf.add_task( p_task=t1c )

# 2.2.2 Then, we add three further tasks that shall start when their predecessor tasks_
↳ have finished
wf.add_task( p_task=t2a, p_pred_tasks=[t1a] )
wf.add_task( p_task=t2b, p_pred_tasks=[t1b] )
wf.add_task( p_task=t2c, p_pred_tasks=[t1c] )

# 2.2.3 Finally, we add three further tasks that build the end of our task chains
wf.add_task( p_task=t3a, p_pred_tasks=[t2a, t2b, t2c] )
wf.add_task( p_task=t3b, p_pred_tasks=[t2a, t2b, t2c] )
wf.add_task( p_task=t3c, p_pred_tasks=[t2a, t2b, t2c] )

```

(continues on next page)



(continued from previous page)

```

# 3 Run the workflow synchronously
wf.run( p_range=mt.Workflow.C_RANGE_NONE, p_wait=True )
wf.log(Log.C_LOG_TYPE_I, 'Result in shared object:\n', wf.get_so().get_results())

# 4 Clear result list in shared object and wait for next run (for better observation)
wf.get_so().clear_results()
wf.log(Log.C_LOG_TYPE_W, 'Short break for better observation of CPU load in perfmeter')
sleep(pause_sec)

# 5 Run the same workflow asynchronously in multithreading mode
wf.run( p_range=mt.Workflow.C_RANGE_THREAD, p_wait=True)
wf.log(Log.C_LOG_TYPE_I, 'Result in shared object:\n', wf.get_so().get_results())

```

## Results

The howto example logs details of the two runs (workflow synchronously/multithreading). A short break between the workflow runs allows a better observation of CPU load in the system monitor.



## Cross Reference

- *API Reference: Multiprocessing*

### 10.1.3 Layer 2 - Mathematics

#### Howto BF-MATH-001: Dimensions, Spaces and Elements

##### Prerequisites

Please install following packages to run this howto

- Numpy

##### Executable code

```
## -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_math_001_spaces_and_elements.py
## -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-05-28  1.0.0     DA       Creation/Release
## -- 2021-09-11  1.0.0     MRD      Change Header information to match our new library.
## -- 2021-09-23  1.0.1     DA       Adaption to changes in class Element
## -- 2021-12-03  1.0.2     DA       New method copy_append_spaces()
## -- 2022-02-25  1.0.3     SY       Refactoring due to auto generated ID in class.
## -- 2022-12-09  1.1.0     DA       Refactoring due to new restrictions in class Set
## -- 2023-03-02  1.1.1     LSB      Refactoring
## -----
"""
Ver. 1.1.1 (2023-03-02)

This module demonstrates how to create a space and subspaces and to spawn elements.

You will learn:

1. How to use the space, subspace, dimension and elements classes of MLPro in native and
   custom implementations.

2. How to create a Space object and how to add dimensions to the Space.

3. How to create an Element in a defined space, i.e. element with values for number of
   dimensions for the related space.

4. How to change values of an Element object.

5. How to calculate the euclidean distance between two elements of given Space.
"""

from mlpro.bf.various import Log
```

(continues on next page)

(continued from previous page)

```

from mlpro.bf.math import *

## -----
↪ -----
## -----
↪ -----
class MathDemo(Log):

    C_TYPE      = 'Demo'
    C_NAME      = 'Spaces & Elements'

    # Some constants for dimension indices to make it more understandable
    C_POS       = 0
    C_VEL       = 1
    C_ACC       = 2
    C_ANG       = 3
    C_AVEL      = 4
    C_AACC      = 5

## -----
↪ -----
    def __init__(self, p_logging=True):
        super().__init__(p_logging=p_logging)
        self.create_euclidian_space()
        self.create_subspace1()
        self.create_subspace2()
        self.create_subspace3()
        self.create_element()
        self.change_elem_values()
        self.calculate_distance()

## -----
↪ -----
    def create_euclidian_space(self):
        self.space = ESpace()
        self.space.add_dim(Dimension( p_name_short='Pos', p_name_long='Position', p_
↪ unit='m', p_unit_latex='m', p_boundaries=[0,100]))
        self.space.add_dim(Dimension( p_name_short='Vel', p_name_long='Velocity', p_
↪ unit='m/s', p_unit_latex='\frac{m}{s}', p_boundaries=[-100,100]))
        self.space.add_dim(Dimension( p_name_short='Acc', p_name_long='Acceleration', p_
↪ unit='m/qs', p_unit_latex='\frac{m}{s^2}', p_boundaries=[-100,100]))
        self.space.add_dim(Dimension( p_name_short='Ang', p_name_long='Angle', p_unit=
↪ 'deg', p_unit_latex='deg', p_boundaries=[-45,45]))
        self.space.add_dim(Dimension( p_name_short='AVel', p_name_long='Angle Velocity',
↪ p_unit='deg/s', p_unit_latex='\frac{deg}{s}', p_boundaries=[-100,100]))
        self.space.add_dim(Dimension( p_name_short='AAcc', p_name_long='Angle_
↪ Acceleration', p_unit='deg/qs', p_unit_latex='\frac{deg}{s^2}', p_boundaries=[-100,
↪ 100]))

```

(continues on next page)

(continued from previous page)

```

        _ids = self.espace.get_dim_ids()
        self.C_POS      = _ids[0]
        self.C_VEL      = _ids[1]
        self.C_ACC      = _ids[2]
        self.C_ANG      = _ids[3]
        self.C_AVEL     = _ids[4]
        self.C_AACC     = _ids[5]

        self.log(self.C_LOG_TYPE_I, '6-dimensional Euclidian space created')

## -----
↳ -----
    def create_subspace1(self):
        self.subspace1 = self.espace.spawn([self.C_POS, self.C_VEL, self.C_ACC])
        self.log(self.C_LOG_TYPE_I, 'Subspace 1 - Number of dimensions and short name of_
↳second dimension:', self.subspace1.get_num_dim(), self.subspace1.get_dim(self.C_VEL).
↳get_name_short())

## -----
↳ -----
    def create_subspace2(self):
        self.subspace2 = self.espace.spawn([self.C_ANG, self.C_AVEL, self.C_AACC])
        self.log(self.C_LOG_TYPE_I, 'Subspace 2 - Number of dimensions and short name of_
↳third dimension:', self.subspace2.get_num_dim(), self.subspace2.get_dim(self.C_AACC).
↳get_name_short())

## -----
↳ -----
    def create_subspace3(self):
        self.subspace3 = self.espace.spawn([self.C_POS, self.C_ANG])
        self.log(self.C_LOG_TYPE_I, 'Subspace 3 - Number of dimensions and short name of_
↳second dimension:', self.subspace3.get_num_dim(), self.subspace3.get_dim(self.C_ANG).
↳get_name_short())

## -----
↳ -----
    def create_element(self):
        self.elem = Element(self.espace)
        self.log(self.C_LOG_TYPE_I, 'New element created with dim ids / values:', self.
↳elem.get_dim_ids(), ' / ', self.elem.get_values())

## -----
↳ -----
    def change_elem_values(self):
        # Changing a value indexed by a unique dimension index...
        self.elem.set_value(self.C_POS, 4.77)
        self.elem.set_value(self.C_VEL, -8.22)

```

(continues on next page)

(continued from previous page)

```

self.log(self.C_LOG_TYPE_I, 'Element changed to ', self.elem.get_values())

## -----
↪ -----
def calculate_distance(self):
    e1 = Element(self.espace)
    e2 = Element(self.espace)
    e2.set_value(self.C_AACC, 1)
    self.log(self.C_LOG_TYPE_I, 'New element e1 =', e1.get_values())
    self.log(self.C_LOG_TYPE_I, 'New element e2 =', e2.get_values())
    self.log(self.C_LOG_TYPE_I, 'Euclidian distance between e1 and e2 =', self.
↪ espace.distance(e1, e2))

if __name__ == "__main__":
    demo = MathDemo()

```

## Results

```

2023-02-10 16:00:33.045206 I Demo "Spaces & Elements": Instantiated
2023-02-10 16:00:33.046206 I Demo "Spaces & Elements": 6-dimensional Euclidian space created
2023-02-10 16:00:33.046206 I Demo "Spaces & Elements": Subspace 1 - Number of dimensions and short name of second dimension: 3 Vel
2023-02-10 16:00:33.046206 I Demo "Spaces & Elements": Subspace 2 - Number of dimensions and short name of third dimension: 3 AAcc
2023-02-10 16:00:33.046206 I Demo "Spaces & Elements": Subspace 3 - Number of dimensions and short name of second dimension: 2 Ang
2023-02-10 16:00:33.046206 I Demo "Spaces & Elements": New element created with dim ids / values: ['f593bb88-1dc4-48a5-b5a2-dc7a55ef7c20', '891247bf-1539-4b10-b91c-ac2832b15a86']
2023-02-10 16:00:33.046206 I Demo "Spaces & Elements": Element changed to [ 4.77 -8.22  0.    0.    0. ]
2023-02-10 16:00:33.046206 I Demo "Spaces & Elements": New element e1 = [0. 0. 0. 0. 0. 0.]
2023-02-10 16:00:33.047204 I Demo "Spaces & Elements": New element e2 = [0. 0. 0. 0. 0. 1.]
2023-02-10 16:00:33.047204 I Demo "Spaces & Elements": Euclidian distance between e1 and e2 = 1.0

```

## Cross Reference

- *API Reference: Math*

## Howto BF-MATH-010: Normalizers

### Prerequisites

Please install following packages to run this howto

- Numpy

### Executable code

```

## -----
↪ -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_math_010_normalizers.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2022-09-16  0.0.0     LSB        Creation
## -- 2022-09-25  1.0.0     LSB        Release of first version
## -- 2022-10-01  1.0.1     LSB        Renormalization
## -- 2022-10-06  1.0.2     LSB        Refactoring

```

(continues on next page)

(continued from previous page)

```

## -- 2022-10-16  1.0.3      LSB      Updating z-transform parameters based on a new data/
↪element(np.ndarray)
## -- 2022-11-03  1.0.4      LSB      refactoring for update with replaced data (Z-
## -- 2023-09-23  1.0.5      LSB      Bug Fix, the input to normalizer shall be copied as ↪
↪it returns the same object
## -----
↪-----

"""
Ver. 1.0.5 (2023-09-23)
Example file for demonstrating the use of MLPro's normalizer for normalizing and de-
↪normalizing data.

You will learn:

1. How to update parameters for Normalization

2. How to update parameters based on single element/data

3. How to normalize a data element (ndarray/mlpro element) by MinMax or ZTransofrm

4. How to denormalize a data element (ndarray/mlpro element) by MinMax or ZTransofrm

5. How to renormalize the data element (ndarray/mlpro element) with respect to the ↪
↪changed parameters
"""
import numpy as np

from mlpro.bf.math.normalizers import *

# checking for internal unit tests
if __name__ == '__main__':
    p_printing = True
else:
    p_printing = False

# Creating Numpy dummy Dataset
my_dataset = np.asarray(((45,-7,65,-87],[21.3,47.1,-41.02,89],[0.12,98.11,11,-56.01],[7.
↪12,55.01,4.78,5.3],[
↪-44.371,-0.521,14.12,8.5],[77.13,-23.14,-7.54,12.32],[8.1,27.61,-
↪31.01,17.8],[
↪4.22,-84.21,47.12,82.11],[-53.22,1.024,5.044,71.23],[9.4,-4.39,
↪12.51,83.01]))

# Creating a dummy set with dummy dimensions

```

(continues on next page)

(continued from previous page)

```

my_set = Set()
my_set.add_dim(Dimension(p_name_short='1', p_boundaries=[10,19]))
my_set.add_dim(Dimension(p_name_short='2', p_boundaries=[-9,10]))

# Creating a dummy element to normalize
my_state = Element(my_set)
my_state.set_values([19,8])

# Creating Normalizer object
my_normalizer_minmax = NormalizerMinMax()
my_normalizer_ztrans = NormalizerZTrans()

# 1. Setting parameters for NormalizationZTrans
my_normalizer_ztrans.update_parameters(p_dataset = my_dataset)
if p_printing:
    print('01. Parameters updated for the Z transformer\n\n')

# 2. Normalizing a numpy array/ a dataset (as an array) in Z transformation
normalized_data = my_normalizer_ztrans.normalize(p_data=my_dataset)
if p_printing:
    print('02. Normalized value(Z transformer):\n', normalized_data,'\n\n')

# 3. De-normalizing a numpy array/ a dataset (as an array) in Z transformation
denormalized_data = my_normalizer_ztrans.denormalize(p_data=normalized_data)
if p_printing:
    print('03. Denormalized value (Z transformer):\n', denormalized_data,'\n\n')

# 4. Updating the parameters using a new element to the dataset
new_data = np.asarray([12,-71,74,-12], dtype=np.float64).reshape(1,4)
my_normalizer_ztrans.update_parameters(p_data_new = new_data)
if p_printing:
    print('04. Parameters updated for the Z transformer\n\n')

# 5. Normalizing the new element with new parameters
normalize_new = my_normalizer_ztrans.normalize(new_data)
if p_printing:
    print('\n05. Normalized Data (Z transformer):', normalize_new,'\n\n')

# 6. Validating the changed parameters
# 6.1 Adding the new element in the dataset
my_dataset = np.append(my_dataset, new_data, axis=0)
# 6.2 Setting up the parameters based on the new dataset
my_normalizer_ztrans.update_parameters(p_dataset=my_dataset)
# 6.3 Normalizing the element for validation

```

(continues on next page)

(continued from previous page)

```

normalized_val = my_normalizer_ztrans.normalize(new_data)
if p_printing:
    print('\n06. Normalized Data (validation Z transformer): ', normalized_val, '\n\n')

# 7. Updating parameters with replaced data

p_data_old = my_dataset[1].copy()
my_dataset[1] = np.asarray([4.1,7.8,-41, 15.3], dtype=np.float64).reshape(1,4)
my_normalizer_ztrans.update_parameters(p_data_new=np.asarray([4.1,7.8,-41, 15.3],
↳dtype=np.float64).reshape(1,4),
                                     p_data_del=p_data_old)
if p_printing:
    print('\n07. Normalization parameters updated for z-transformer based on replaced_
↳data')

# 8. Normalizing the new element with new parameters
normalize_new = my_normalizer_ztrans.normalize(new_data)
if p_printing:
    print('\n08. Normalized Data (Z transformer):', normalize_new, '\n\n')

# 9. Validating the updated parameters
# 9.1. Updating parameters based on the new dataset
my_normalizer_ztrans.update_parameters(p_dataset=my_dataset)
normalized_val = my_normalizer_ztrans.normalize(new_data)
if p_printing:
    print('\n09. Normalized Data (validation Z transformer): ', normalized_val, '\n\n')

# 10. Setting parameters for Normalization
my_normalizer_minmax.update_parameters(p_set = my_set)
if p_printing:
    print('10. Parameters updated for the MinMax Normalizer\n\n')

# 11. Normalizing using MinMax
normalized_state = my_normalizer_minmax.normalize(my_state.copy())
if p_printing:
    print('11. Normalized value (MinMax Normalizer):\n', normalized_state.get_values(), '\n
↳n\n')

# 12. De-normalizing using MinMax
denormalized_state = my_normalizer_minmax.denormalize(normalized_state.copy())
if p_printing:
    print('12. Denormalized value (MinMax Normalizer):\n', denormalized_state.get_
↳values(), '\n\n')

```

(continues on next page)



(continued from previous page)

```

# 13. Updating the boundaries of the dimension
my_set.get_dim(p_id=my_set.get_dim_ids()[0]).set_boundaries([-10,51])
my_set.get_dim(p_id=my_set.get_dim_ids()[1]).set_boundaries([-5,10])
if p_printing:
    print('13. Boundaries updated (MinMax Normalizer)\n\n')

# 14. Updating the normalization parameters for the new set
my_normalizer_minmax.update_parameters(my_set)
if p_printing:
    print('14. Parameters updated for MinMax normalizer\n\n')

# 15. Renormalizing the previously normalized data with the new parameters
re_normalized_state = my_normalizer_minmax.renormalize(normalized_state.copy())
if p_printing:
    print('15. Renormalized value (MinMax Normalizer):\n', re_normalized_state.get_
↪values(), '\n\n')

# 16. Validating the renormalization
normalized_state = my_normalizer_minmax.normalize(my_state.copy())
if p_printing:
    print('16. Normalized value (Validation renormalization):\n', normalized_state.get_
↪values(), '\n\n')

```

## Results

The results will be available as follows

```

01. Parameters updated for the Z transformer

02. Normalized value(Z transformer):
[[ 1.04497494 -0.38178705  1.89666962 -1.91464488]
 [ 0.38490458  0.7682956  -1.63116063  1.15923813]
 [-0.20498109  1.85268962  0.09981211 -1.37339696]
 [-0.0100236   0.93645002 -0.10715926 -0.30260282]
 [-1.44410306 -0.24405349  0.20363054 -0.24671404]
 [ 1.93982982 -0.72489859 -0.51710897 -0.17999681]
 [ 0.01727045  0.35396823 -1.29807649 -0.08428727]
 [-0.0907917  -2.0231527  1.30171013  1.0389026 ]
 [-1.69055718 -0.21120917 -0.09837462  0.84888074]
 [ 0.05347684 -0.32630247  0.15005757  1.05462132]]

03. Denormalized value (Z transformer):
[[ 45.    -7.    65.   -87.   ]
 [ 21.3  47.1  -41.02  89.   ]
 [  0.12  98.11  11.    -56.01 ]
 [  7.12  55.01   4.78   5.3  ]
 [-44.371 -0.521  14.12   8.5  ]

```

(continues on next page)

(continued from previous page)

```
[ 77.13  -23.14  -7.54  12.32 ]
[  8.1   27.61 -31.01  17.8  ]
[  4.22 -84.21  47.12  82.11 ]
[-53.22   1.024  5.044  71.23 ]
[  9.4   -4.39  12.51  83.01 ]]
```

04. Parameters updated for the Z transformer

05. Normalized Data (Z transformer): [[ 0.11994467 -1.47066196 1.74588644 -0.56725513]]

06. Normalized Data (validation Z transformer): [[ 0.11994467 -1.47066196 1.74588644 -  
↪0.56725513]]

07. Normalization parameters updated for z-transformer based on replaced data

08. Normalized Data (Z transformer): [[ 0.16683367 -1.45314853 1.7459814 -0.48625054]]

09. Normalized Data (validation Z transformer): [[ 0.16683367 -1.45314853 1.7459814 -  
↪0.48625054]]

10. Parameters updated for the MinMax Normalizer

11. Normalized value (MinMax Normalizer):  
[1. 0.78947368]

12. Denormalized value (MinMax Normalizer):  
[19. 8.]

13. Boundaries updated (MinMax Normalizer)

14. Parameters updated for MinMax normalizer

15. Renormalized value (MinMax Normalizer):  
[2.60655738 9.86666667]

16. Normalized value (Validation renormalization):

(continues on next page)

(continued from previous page)

[-0.58667025 0.98222222]

**Cross Reference**

- *API Reference: Normalizers*

**10.1.4 Layer 3 - Application Support****Stream Processing****Howto BF-STREAMS-001: Accessing Native Data From MLPro****Executable code**

```
## -----
↪ -----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪ learning
## -- Module  : howto_bf_streams_001_accessing_native_data_from_mlpro.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-11-08  1.0.0     DA       Creation
## -- 2022-12-14  1.0.1     DA       Corrections
## -- 2023-02-02  1.0.2     DA       Correction of time measurement
## -----
↪ -----

"""
Ver. 1.0.2 (2023-02-02)

This module demonstrates the basic use of native generic data streams provided by MLPro.
↪ To this regard,
all data streams of the related provider class will be determined and iterated.

You will learn:

1) How to access MLPro's native data streams.

2) How to iterate the instances of a native stream.

3) How to access feature data of a native stream.

"""

from datetime import datetime
from mlpro.bf.streams.streams import *
from mlpro.bf.various import Log
```

(continues on next page)

(continued from previous page)

```

# 0 Prepare Demo/Unit test mode
if __name__ == '__main__':
    logging = Log.C_LOG_ALL
else:
    logging = Log.C_LOG_NOTHING

# 1 Create a Wrapper for OpenML stream provider
mlpro = StreamProviderMLPro(p_logging=logging)

# 2 Determine native data streams provided by MLPro
for stream in mlpro.get_stream_list( p_logging=logging ):
    stream.switch_logging( p_logging=logging )
    try:
        labels = stream.get_label_space().get_num_dim()
    except:
        labels = 0

    stream.log(Log.C_LOG_TYPE_W, 'Features:', stream.get_feature_space().get_num_dim(),
    ↪ ', Labels:', labels, ', Instances:', stream.get_num_instances() )

if __name__ == '__main__':
    input('\nPress ENTER to iterate all streams dark...\n')

# 3 Performance test: iterate all data streams dark and measure the time
for stream in mlpro.get_stream_list( p_logging=logging ):
    stream.switch_logging( p_logging=logging )
    stream.log(Log.C_LOG_TYPE_W, 'Number of instances:', stream.get_num_instances() )
    stream.switch_logging( p_logging=Log.C_LOG_NOTHING )

    # 3.1 Iterate all instances of the stream
    tp_start = datetime.now()
    myiterator = iter(stream)
    for i, curr_instance in enumerate(myiterator):
        curr_data = curr_instance.get_feature_data().get_values()

    tp_end = datetime.now()
    duration = tp_end - tp_start
    duration_sec = ( duration.seconds * 1000000 + duration.microseconds + 1 ) / 1000000
    rate = myiterator.get_num_instances() / duration_sec

    myiterator.switch_logging( p_logging=logging )
    myiterator.log(Log.C_LOG_TYPE_W, 'Done in', round(duration_sec,3), ' seconds_
    ↪ (throughput =', round(rate), 'instances/sec')

```

## Results

```

2024-02-09 12:00:29.748098 I Stream Provider "MLPro": Instantiated
2024-02-09 12:00:29.748098 I Stream "Random 100 x 1000": Instantiated
2024-02-09 12:00:29.748098 I Stream "Double Spiral 2D x 721": Instantiated
2024-02-09 12:00:29.748098 I Benchmark "Clouds N-Dim": Instantiated
2024-02-09 12:00:29.748098 I Benchmark "Point Outliers N-Dim": Instantiated
2024-02-09 12:00:29.748098 I Benchmark "Static Clouds 2D": Instantiated
2024-02-09 12:00:29.748098 I Benchmark "Static Clouds 3D": Instantiated
2024-02-09 12:00:29.748098 I Benchmark "Dynamic Clouds 2D": Instantiated
2024-02-09 12:00:29.748098 I Benchmark "Dynamic Clouds 3D": Instantiated
2024-02-09 12:00:29.748098 I Stream Provider "MLPro": Getting list of streams...
2024-02-09 12:00:29.748098 I Stream Provider "MLPro": Number of streams found: 8
2024-02-09 12:00:29.748098 W Stream "Random 100 x 1000": Features: 10 , Labels: 2 , Instances: 1000
2024-02-09 12:00:29.748098 W Stream "Double Spiral 2D x 721": Features: 2 , Labels: 0 , Instances: 721
2024-02-09 12:00:29.748098 W Benchmark "Clouds N-Dim": Features: 3 , Labels: 0 , Instances: 1000
2024-02-09 12:00:29.748098 W Benchmark "Point Outliers N-Dim": Features: 4 , Labels: 0 , Instances: 1000
2024-02-09 12:00:29.748098 W Benchmark "Static Clouds 2D": Features: 2 , Labels: 0 , Instances: 1000
2024-02-09 12:00:29.748098 W Benchmark "Static Clouds 3D": Features: 3 , Labels: 0 , Instances: 2000
2024-02-09 12:00:29.748098 W Benchmark "Dynamic Clouds 2D": Features: 2 , Labels: 0 , Instances: 5000
2024-02-09 12:00:29.748098 W Benchmark "Dynamic Clouds 3D": Features: 3 , Labels: 0 , Instances: 10000

Press ENTER to iterate all streams dark...

2024-02-09 12:00:31.848824 I Stream Provider "MLPro": Getting list of streams...
2024-02-09 12:00:31.848824 I Stream Provider "MLPro": Number of streams found: 8
2024-02-09 12:00:31.864458 W Stream "Random 100 x 1000": Number of instances: 1000
2024-02-09 12:00:31.871080 W Stream "Random 100 x 1000": Done in 0.007 seconds (throughput = 150989 instances/sec)
2024-02-09 12:00:31.871080 W Stream "Double Spiral 2D x 721": Number of instances: 721
2024-02-09 12:00:31.876690 W Stream "Double Spiral 2D x 721": Done in 0.006 seconds (throughput = 128498 instances/sec)
2024-02-09 12:00:31.877055 W Benchmark "Clouds N-Dim": Number of instances: 1000
2024-02-09 12:00:31.877055 W Benchmark "Clouds N-Dim": Done in 0.0 seconds (throughput = 1000000000 instances/sec)
2024-02-09 12:00:31.877055 W Benchmark "Point Outliers N-Dim": Number of instances: 1000
2024-02-09 12:00:31.895931 W Benchmark "Point Outliers N-Dim": Done in 0.019 seconds (throughput = 52975 instances/sec)
2024-02-09 12:00:31.895931 W Benchmark "Static Clouds 2D": Number of instances: 1000
2024-02-09 12:00:31.895931 W Benchmark "Static Clouds 2D": Done in 0.0 seconds (throughput = 1000000000 instances/sec)
2024-02-09 12:00:31.895931 W Benchmark "Static Clouds 3D": Number of instances: 2000
2024-02-09 12:00:31.910516 W Benchmark "Static Clouds 3D": Done in 0.015 seconds (throughput = 137118 instances/sec)
2024-02-09 12:00:31.910516 W Benchmark "Dynamic Clouds 2D": Number of instances: 5000
2024-02-09 12:00:31.943169 W Benchmark "Dynamic Clouds 2D": Done in 0.033 seconds (throughput = 153121 instances/sec)
2024-02-09 12:00:31.943169 W Benchmark "Dynamic Clouds 3D": Number of instances: 10000
2024-02-09 12:00:32.009285 W Benchmark "Dynamic Clouds 3D": Done in 0.066 seconds (throughput = 151247 instances/sec)

```

## Cross Reference

- [API Reference: Streams](#)

## Howto BF-STREAMS-002: Accessing Data From CSV Files

### Executable code

```

## -----
↪ -----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪ learning
## -- Module  : howto_bf_streams_002_accessing_data_from_csv_files.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2023-03-03  0.0.0     SY         Creation
## -- 2023-03-03  1.0.0     SY         First release
## -----
↪ -----
"""

```

(continues on next page)

(continued from previous page)

Ver. 1.0.0 (2023-03-03)

This module demonstrates loading and converting data stored in csv files to be  
 ↳ compatible for data  
 streams provided by MLPro.

You will learn:

- 1) How to load and convert data from csv files.
- 2) How to iterate the instances of a native stream.
- 3) How to access feature data of a native stream.

"""

```
from datetime import datetime
from mlpro.bf.streams.streams import *
from mlpro.bf.various import *
from mlpro.bf.data import *
import random
from pathlib import Path
```

```
# 0 Prepare Demo/Unit test mode
if __name__ == '__main__':
    logging = Log.C_LOG_ALL
```

```
# 1 Generate random data and store them in csv format
```

```
num_eps = 10
num_cycles = 10000
data_names = ["action", "states_1", "states_2", "model_loss"]
data_printing = {"action": [True, 0, 10],
                  "states_1": [True, 0, 4],
                  "states_2": [True, 0, 4],
                  "model_loss": [True, 0, -1]}
```

```
mem = DataStoring(data_names)
for ep in range(num_eps):
    ep_id = ("ep. %s"%str(ep+1))
    mem.add_frame(ep_id)
    for i in range(num_cycles):
        mem.memorize("action", ep_id, random.uniform(0+(ep*0.5), 5+(ep*0.5)))
        mem.memorize("states_1", ep_id, random.uniform(2-(ep*0.2), 4-(ep*0.2)))
        mem.memorize("states_2", ep_id, random.uniform(0+(ep*0.2), 2+(ep*0.2)))
        mem.memorize("model_loss", ep_id, random.uniform(0.25-(ep*0.02), 1-(ep*0.07)))
```

```
path_save = str(Path.home())
mem.save_data(path_save, "data_storage", "\t")
```

(continues on next page)

(continued from previous page)

```

# 2 Instantiate Stream
stream = StreamMLProCSV(p_logging=logging,
                        p_path_load=path_save,
                        p_csv_filename="data_storage.csv",
                        p_delimiter="\t",
                        p_frame=True,
                        p_header=True,
                        p_list_features=["action", "states_1", "states_2"],
                        p_list_labels=["model_loss"])

# 3. load data from the csv file
data_names = []
mem_from_csv = DataStoring(data_names)
mem_from_csv.load_data(path_save, "data_storage.csv", "\t")

# 4 Performance test: iterate all data streams dark and measure the time
input('\nPress ENTER to iterate all streams dark...\n')

# 4.1 Iterate all instances of the stream
tp_start = datetime.now()
myiterator = iter(stream)

stream.switch_logging( p_logging=logging )
try:
    labels = stream.get_label_space().get_num_dim()
except:
    labels = 0
stream.log(Log.C_LOG_TYPE_W, 'Features:', stream.get_feature_space().get_num_dim(),
↳ ', Labels:', labels, ', Instances:', stream.get_num_instances() )

for i, curr_instance in enumerate(myiterator):
    curr_data = curr_instance.get_feature_data().get_values()

tp_end      = datetime.now()
duration    = tp_end - tp_start
duration_sec = ( duration.seconds * 1000000 + duration.microseconds + 1 ) / 1000000
rate        = myiterator.get_num_instances() / duration_sec

myiterator.switch_logging( p_logging=logging )
myiterator.log(Log.C_LOG_TYPE_W, 'Done in', round(duration_sec,3), ' seconds_
↳ (throughput =', round(rate), 'instances/sec)')

```

## Results

```
YYYY-MM-DD HH:MM:SS.SSSSSS I Stream "": Instantiated
```

```
Press ENTER to iterate all streams dark...
```

(continues on next page)

(continued from previous page)

```

YYYY-MM-DD HH:MM:SS.SSSSSS I Stream "": Reset
YYYY-MM-DD HH:MM:SS.SSSSSS W Stream "": Features: 3 , Labels: 1 , Instances: 100000
YYYY-MM-DD HH:MM:SS.SSSSSS I Stream "": Reset
YYYY-MM-DD HH:MM:SS.SSSSSS W Stream "": Done in 1.172 seconds (throughput = 85336
↪instances/sec)

```

### Cross Reference

- API Reference: Data from CSV files

## Howto BF-STREAMS-003: Visualizing 10-dimensional Random Stream Provided By MLPro

### Executable code

```

## -----
↪-----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪learning
## -- Module  : howto_bf_streams_003_native_stream_Rnd10Dx1000.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd Ver.      Auth.      Description
## -- 2024-02-06 1.0.0     DA          Creation/First implementation
## -----
↪-----

"""
Ver. 1.0.0 (2024-02-06)

This module demonstrates and visualizes the native stream Rnd10Dx1000 which generates_
↪1000
10-dimensional random instances.

You will learn:

1) The properties and use of native stream Rnd10Dx1000.

2) How to set up a stream workflow without a stream task.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with default visualization.

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.various import Log

```

(continues on next page)



(continued from previous page)

```

## -----
↪ -----
## -----
↪ -----
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See
    ↪ class
    ↪ mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'My stream scenario'

## -----
↪ -----
    def _setup(self, p_mode, p_visualize:bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_seed=1, p_logging=p_logging)
        stream = provider_mlpro.get_stream('Rnd10Dx1000', p_mode=p_mode, p_logging=p_
↪ logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=StreamWorkflow.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 3 Return stream and workflow
        return stream, workflow

# 1 Preparation of demo/unit test mode
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    cycle_limit = 200
    logging      = Log.C_LOG_ALL
    visualize    = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False

```

(continues on next page)

(continued from previous page)

```

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                        p_cycle_limit=cycle_limit,
                        p_visualize=visualize,
                        p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot()
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')

```

## Results

## Cross Reference

- *API Reference: 10D Random Stream*
- *API Reference: Streams*

## Howto BF-STREAMS-004: Visualizing 2D Double Spiral Stream Provided By MLPro

## Executable code

```

## -----
↪-----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪learning
## -- Module  : howto_bf_streams_004_native_stream_DoubleSpiral2D.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2024-02-06  1.0.0     DA         Creation/First implementation
## -----
↪-----
"""
Ver. 1.0.0 (2024-02-06)

This module demonstrates and visualizes the native stream DoubleSpiral2D which generates_
↪721 instances
shaping a 2-dimensional double spiral.

You will learn:

```

(continues on next page)

(continued from previous page)

```

1) The properties and use of native stream DoubleSpiral2D.
2) How to set up a stream workflow without a stream task.
3) How to set up a stream scenario based on a stream and a processing stream workflow.
4) How to run a stream scenario dark or with default visualization.

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.various import Log

## -----
↪ -----
## -----
↪ -----
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See
    ↪ class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'My stream scenario'

## -----
↪ -----
    def _setup(self, p_mode, p_visualize:bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_seed=1, p_logging=p_logging)
        stream = provider_mlpro.get_stream('DoubleSpiral2D', p_mode=p_mode, p_logging=p_
        ↪ logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=StreamWorkflow.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 3 Return stream and workflow
        return stream, workflow

```

(continues on next page)

(continued from previous page)

```
# 1 Preparation of demo/unit test mode
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    cycle_limit = 720
    logging      = Log.C_LOG_ALL
    visualize    = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                        p_cycle_limit=cycle_limit,
                        p_visualize=visualize,
                        p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot()
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

## Results

## Cross Reference

- *API Reference: Double Spiral Stream*
- *API Reference: Streams*

## Howto BF-STREAMS-005: Visualizing Multivariate Point Outlier Stream Provided By MLPro

### Executable code

```
## -----
↪-----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪learning
## -- Module  : howto_bf_streams_005_native_stream_PointOutliersND.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2024-02-06  1.0.0     DA         Creation/First implementation
## -- 2024-04-26  1.1.0     DA         Refactoring
## -----
↪-----

"""
Ver. 1.1.0 (2024-04-26)

This module demonstrates and visualizes the native stream PointOutliersND which_
↪generates an infinite
instances number of n-dimensional instances. Each feature is based on a configurable_
↪baseline function.
Additionally, random point outliers are induced.

You will learn:

1) The properties and use of native stream PointOutliersND.

2) How to set up a stream workflow without a stream task.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with default visualization.

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.various import Log

## -----
↪-----
## -----
↪-----
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See_
    (continues on next page)
```

(continued from previous page)

```

↪ class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'My stream scenario'

## -----
↪ -----
    def _setup(self, p_mode, p_visualize:bool, p_logging):

        # 1 Import a native stream from MLPro
        stream = StreamMLProPOutliers( p_functions = ['sin', 'cos', 'const'],
                                       p_outlier_rate = 0.01,
                                       p_visualize = p_visualize,
                                       p_logging = p_logging )

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name = 'wf1',
                                   p_range_max = StreamWorkflow.C_RANGE_NONE,
                                   p_visualize = p_visualize,
                                   p_logging = logging )

        # 3 Return stream and workflow
        return stream, workflow

# 1 Preparation of demo/unit test mode
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    cycle_limit = 720
    logging      = Log.C_LOG_ALL
    visualize    = True
    step_rate    = 2

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False
    step_rate    = 1

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                        p_cycle_limit=cycle_limit,
                        p_visualize=visualize,
                        p_logging=logging )

```

(continues on next page)

(continued from previous page)

```
# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                         p_view_autoselect = False,
                                                         p_step_rate = step_rate ) )

    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

## Results

## Cross Reference

- *API Reference: Multivariate Point Outlier Stream*
- *API Reference: Streams*

## Howto BF-STREAMS-006: Visualizing Static 2D Random Point Clouds Provided By MLPro

## Executable code

```
## -----
↪-----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪learning
## -- Module  : howto_bf_streams_006_Clouds2D4C1000Static.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2024-02-06  1.0.0     DA         Creation/First implementation
## -----
↪-----

"""
Ver. 1.0.0 (2024-02-06)

This module demonstrates and visualizes the native stream Clouds2D4C1000Static which_
↪generates 1000
2-dimensional random instances that form 4 point clouds.

You will learn:

1) The properties and use of native stream Clouds2D4C1000Static.
```

(continues on next page)

(continued from previous page)

- 2) How to set up a stream workflow without a stream task.
- 3) How to set up a stream scenario based on a stream and a processing stream workflow.
- 4) How to run a stream scenario dark or with default visualization.

```

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.various import Log

## -----
↪ -----
## -----
↪ -----
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See ↪
    ↪ class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'My stream scenario'

## -----
↪ -----
    def _setup(self, p_mode, p_visualize:bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_seed=1, p_logging=p_logging)
        stream = provider_mlpro.get_stream('Clouds2D4C1000Static', p_mode=p_mode, p_
↪ logging=p_logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=StreamWorkflow.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 3 Return stream and workflow
        return stream, workflow

```

(continues on next page)



(continued from previous page)

```

# 1 Preparation of demo/unit test mode
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    cycle_limit = 500
    logging      = Log.C_LOG_ALL
    visualize    = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                          p_cycle_limit=cycle_limit,
                          p_visualize=visualize,
                          p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot()
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')

```

## Results

## Cross Reference

- *API Reference: Random Point Clouds*
- *API Reference: Streams*

## Howto BF-STREAMS-007: Visualizing Dynamic 2D Random Point Clouds Provided By MLPro

## Executable code

```

## -----
↪ -----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪ learning
## -- Module  : howto_bf_streams_007_native_stream_Clouds2D4C5000Dynamic.py
## -----
↪ -----

```

(continues on next page)

(continued from previous page)

```

## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2024-02-06  1.0.0    DA          Creation/First implementation
## -----
↪ -----

"""
Ver. 1.0.0 (2024-02-06)

This module demonstrates and visualizes the native stream Clouds2D4C5000Dynamic which
↪ generates 5000
2-dimensional random instances that form 4 moving point clouds.

You will learn:

1) The properties and use of native stream Clouds2D4C5000Dynamic.
2) How to set up a stream workflow without a stream task.
3) How to set up a stream scenario based on a stream and a processing stream workflow.
4) How to run a stream scenario dark or with default visualization.

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.various import Log

## -----
↪ -----
## -----
↪ -----

class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See
    ↪ class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'My stream scenario'

## -----
↪ -----

    def _setup(self, p_mode, p_visualize:bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_seed=1, p_logging=p_logging)
        stream = provider_mlpro.get_stream('Clouds2D4C5000Dynamic', p_mode=p_mode, p_

```

(continues on next page)

(continued from previous page)

```

↪ logging=p_logging)

    # 2 Set up a stream workflow
    workflow = StreamWorkflow( p_name='wf1',
                               p_range_max=StreamWorkflow.C_RANGE_NONE,
                               p_visualize=p_visualize,
                               p_logging=logging )

    # 3 Return stream and workflow
    return stream, workflow

# 1 Preparation of demo/unit test mode
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    cycle_limit = 1000
    logging      = Log.C_LOG_ALL
    visualize    = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                          p_cycle_limit=cycle_limit,
                          p_visualize=visualize,
                          p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot()
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')

```

**Results****Cross Reference**

- *API Reference: Random Point Clouds*
- *API Reference: Streams*

## Howto BF-STREAMS-008: Visualizing Static 3D Random Point Clouds Provided By MLPro

### Executable code

```
## -----
↪ -----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪ learning
## -- Module  : howto_bf_streams_008_native_stream_Clouds3D8C2000Static.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2024-02-06  1.0.0     DA         Creation/First implementation
## -----
↪ -----

"""
Ver. 1.0.0 (2024-02-06)

This module demonstrates and visualizes the native stream Clouds3D8C2000Static which_
↪ generates 2000
3-dimensional random instances that form 8 point clouds.

You will learn:

1) The properties and use of native stream Clouds3D8C2000Static.

2) How to set up a stream workflow without a stream task.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with default visualization.

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.various import Log

## -----
↪ -----
## -----
↪ -----

class MyScenario (StreamScenario):
    """
```

(continues on next page)

(continued from previous page)

```

    Example of a custom stream scenario including a stream and a stream workflow. See
    ↪class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'My stream scenario'

    ## -----
    ↪-----
    def _setup(self, p_mode, p_visualize:bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_seed=1, p_logging=p_logging)
        stream = provider_mlpro.get_stream('Clouds3D8C2000Static', p_mode=p_mode, p_
    ↪logging=p_logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=StreamWorkflow.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 3 Return stream and workflow
        return stream, workflow

# 1 Preparation of demo/unit test mode
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    cycle_limit = 2000
    logging      = Log.C_LOG_ALL
    visualize    = True
    step_rate    = 10

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False
    step_rate    = 1

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                          p_cycle_limit=cycle_limit,
                          p_visualize=visualize,
                          p_logging=logging )

```

(continues on next page)

(continued from previous page)

```
# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                         p_view_autoselect = True,
                                                         p_step_rate = step_rate ) )

    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

## Results

### Cross Reference

- *API Reference: Random Point Clouds*
- *API Reference: Streams*

## Howto BF-STREAMS-009: Visualizing Dynamic 3D Random Point Clouds Provided By MLPro

### Executable code

```
## -----
↪-----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪learning
## -- Module  : howto_bf_streams_009_native_stream_Clouds3D8C10000Dynamic.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2024-02-06  1.0.0     DA         Creation/First implementation
## -----
↪-----
"""
Ver. 1.0.0 (2024-02-06)

This module demonstrates and visualizes the native stream Clouds3D8C2000Static which_
↪generates 10000
3-dimensional random instances that form 8 moving point clouds.

You will learn:

1) The properties and use of native stream Clouds3D8C10000Dynamic.
```

(continues on next page)

(continued from previous page)

- 2) How to set up a stream workflow without a stream task.
- 3) How to set up a stream scenario based on a stream and a processing stream workflow.
- 4) How to run a stream scenario dark or with default visualization.

```

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.various import Log

## -----
↪ -----
## -----
↪ -----
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See ↪
    ↪ class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'My stream scenario'

## -----
↪ -----
    def _setup(self, p_mode, p_visualize:bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_seed=1, p_logging=p_logging)
        stream = provider_mlpro.get_stream('Clouds3D8C100000Dynamic', p_mode=p_mode, p_
↪ logging=p_logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=StreamWorkflow.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 3 Return stream and workflow
        return stream, workflow

```

(continues on next page)

(continued from previous page)

```

# 1 Preparation of demo/unit test mode
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    cycle_limit = 5000
    logging      = Log.C_LOG_ALL
    visualize    = True
    step_rate    = 50

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False
    step_rate    = 1

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                        p_cycle_limit=cycle_limit,
                        p_visualize=visualize,
                        p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                       p_view_autoselect = True,
                                                       p_step_rate = step_rate ) )

    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')

```

## Results

### Cross Reference

- *API Reference: Random Point Clouds*
- *API Reference: Streams*



## Howto BF-STREAMS-010: Visualizing Multivariate Random Cloud Generator in 3D Mode Provided By MLPro

### Executable code

```
## -----
↪-----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪learning
## -- Module  : howto_bf_streams_010_native_stream_CloudsNDim.py
## -----
↪-----
## -- History :
## -- 2024-02-06  1.0.0      DA      Creation/First implementation
## -----
↪-----

"""
Ver. 1.0.0 (2024-02-06)

This module demonstrates and visualizes the native stream CloudsNDim. It is the freely_
↪configurable
n-dimensional random point cloud generator behind native streams like_
↪Clouds2D4C1000Static, etc.

You will learn:

1) The properties and use of native stream Clouds3D8C10000Dynamic.

2) How to set up a stream workflow without a stream task.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with default visualization.

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.various import Log

## -----
↪-----
## -----
↪-----
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See_
    ↪class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """
```

(continues on next page)

(continued from previous page)

```

"""

C_NAME      = 'My stream scenario'

## -----
↪ -----
def _setup(self, p_mode, p_visualize:bool, p_logging):

    # 1 Import a native stream from MLPro
    stream = StreamMLProClouds( p_num_dim = 3,
                                p_num_instances = 2000,
                                p_num_clouds = 5,
                                p_seed = 1,
                                p_radii = [100, 150, 200, 250, 300],
                                p_weights = [2,3,4,5,6],
                                p_logging=Log.C_LOG_NOTHING )

    # 2 Set up a stream workflow
    workflow = StreamWorkflow( p_name='wf1',
                               p_range_max=StreamWorkflow.C_RANGE_NONE,
                               p_visualize=p_visualize,
                               p_logging=logging )

    # 3 Return stream and workflow
    return stream, workflow

# 1 Preparation of demo/unit test mode
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    cycle_limit = 2000
    logging      = Log.C_LOG_ALL
    visualize    = True
    step_rate    = 5

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False
    step_rate    = 1

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                          p_cycle_limit=cycle_limit,
                          p_visualize=visualize,

```

(continues on next page)

(continued from previous page)

```

        p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                         p_view_autoselect = True,
                                                         p_step_rate = step_rate ) )

    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')

```

## Results

## Cross Reference

- *API Reference: Random Point Clouds*
- *API Reference: Streams*

## Howto BF-STREAMS-020: Streams Sampler

## Executable code

```

## -----
↪-----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪learning
## -- Module  : howto_bf_streams_020_sampler.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-04-10  0.0.0     SY        Creation
## -- 2023-04-14  1.0.0     SY        First release
## -- 2023-04-16  1.0.1     SY        Add more sampler methods to this howto
## -- 2023-04-17  1.1.0     SY        Replace StreamMLProCSV to Rnd10Dx1000
## -----
↪-----
"""
Ver. 1.0.1 (2023-04-17)

This howto file demonstrates the incorporation of a stream sampler in a stream scenario.

You will learn:

```

(continues on next page)

(continued from previous page)

```

1) How to access a MLPro's native data stream.
2) How to iterate the instances of a native stream.
3) How to incorporate a stream sampler.
4) Update a sampler method with another sampler method after instantiation of a stream
"""

from datetime import datetime
from mlpro.bf.streams.streams import *
from mlpro.bf.streams.samplers import *
from mlpro.bf.various import *
from mlpro.bf.data import *

# 0 Prepare Demo/Unit test mode
if __name__ == '__main__':
    logging = Log.C_LOG_ALL
else:
    logging = Log.C_LOG_NOTHING

# 1 Determine a native data stream provided by MLPro
stream = StreamMLProRnd10D( p_logging=logging, p_sampler=SamplerRND(p_max_step_rate=10,
↳p_seed=10) )

if __name__ == '__main__':
    input('\nPress ENTER to iterate all streams dark...\n')

# 2 Iterate all instances of the stream
tp_start = datetime.now()
myiterator = iter(stream)

stream.switch_logging( p_logging=logging )
try:
    labels = stream.get_label_space().get_num_dim()
except:
    labels = 0
stream.log(Log.C_LOG_TYPE_W, 'Features:', stream.get_feature_space().get_num_dim(), ', ↳
↳Labels:', labels, ', Instances:', stream.get_num_instances() )

for i, curr_instance in enumerate(myiterator):
    curr_data = curr_instance.get_feature_data().get_values()

myiterator.switch_logging( p_logging=logging )
stream.log(Log.C_LOG_TYPE_W, 'Number of instances being sampled:', int(i+1) )

```

(continues on next page)

(continued from previous page)

```

tp_end      = datetime.now()
duration    = tp_end - tp_start
duration_sec = ( duration.seconds * 1000000 + duration.microseconds + 1 ) / 1000000
rate        = myiterator.get_num_instances() / duration_sec
myiterator.log(Log.C_LOG_TYPE_W, 'Done in', round(duration_sec,3), ' seconds (throughput_
↳=' , round(rate), 'instances/sec')

# 3 Change the sampler method

# 3.1 Weigthed random sampling
stream._sampler = SamplerWeightedRND(p_threshold=0.75, p_seed=10)

for i, curr_instance in enumerate(myiterator):
    curr_data = curr_instance.get_feature_data().get_values()

myiterator.switch_logging( p_logging=logging )
stream.log(Log.C_LOG_TYPE_W, 'Number of instances being sampled:', int(i+1) )

# 3.2.1 Reservoir sampling with number of instances
stream._sampler = SamplerReservoir(p_num_instances=stream.get_num_instances(), p_
↳reservoir_size=100, p_seed=10)

for i, curr_instance in enumerate(myiterator):
    curr_data = curr_instance.get_feature_data().get_values()

myiterator.switch_logging( p_logging=logging )
stream.log(Log.C_LOG_TYPE_W, 'Number of instances being sampled:', int(i+1) )

# 3.2.2 Reservoir sampling without number of instances
stream._sampler = SamplerReservoir(p_reservoir_size=100, p_seed=10)

for i, curr_instance in enumerate(myiterator):
    curr_data = curr_instance.get_feature_data().get_values()

myiterator.switch_logging( p_logging=logging )
stream.log(Log.C_LOG_TYPE_W, 'Number of instances being sampled:', int(i+1) )

# 3.3 Min-wise sampling
stream._sampler = SamplerMinWise(p_cluster_size=20, p_seed=10)

for i, curr_instance in enumerate(myiterator):
    curr_data = curr_instance.get_feature_data().get_values()

myiterator.switch_logging( p_logging=logging )
stream.log(Log.C_LOG_TYPE_W, 'Number of instances being sampled:', int(i+1) )

```

## Results

```

2024-02-09 16:00:18.341310 I Stream "Random 100 x 1000": Instantiated
Press ENTER to iterate all streams dark...

2024-02-09 16:00:21.568912 I Stream "Random 100 x 1000": Reset
2024-02-09 16:00:21.572307 W Stream "Random 100 x 1000": Features: 10 , Labels: 2 , Instances: 1000
2024-02-09 16:00:21.573106 I Stream "Random 100 x 1000": Reset
2024-02-09 16:00:21.581973 W Stream "Random 100 x 1000": Number of instances being sampled: 150
2024-02-09 16:00:21.581973 W Stream "Random 100 x 1000": Done in 0.013 seconds (throughput = 76558 instances/sec)
2024-02-09 16:00:21.582978 I Stream "Random 100 x 1000": Reset
2024-02-09 16:00:21.586613 W Stream "Random 100 x 1000": Number of instances being sampled: 239
2024-02-09 16:00:21.592937 I Stream "Random 100 x 1000": Reset
2024-02-09 16:00:21.595912 W Stream "Random 100 x 1000": Number of instances being sampled: 100
2024-02-09 16:00:21.595912 I Stream "Random 100 x 1000": Reset
2024-02-09 16:00:21.595912 W Stream "Random 100 x 1000": Number of instances being sampled: 333
2024-02-09 16:00:21.595912 I Stream "Random 100 x 1000": Reset
2024-02-09 16:00:21.611797 W Stream "Random 100 x 1000": Number of instances being sampled: 50

```

## Cross Reference

- *API Reference: Random Samplers*
- *API Reference: Data from CSV files*

## Howto BF-STREAMS-101: Basics of Streams

### Executable code

```

## -----
↪ -----
## -- Project : MLPro - The integrative middleware framework for standardized machine_
↪ learning
## -- Module  : howto_bf_streams_101_basics.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2022-10-27  0.0.0     DA         Creation
## -- 2022-12-14  1.0.0     DA         First implementation
## -- 2024-02-06  1.1.0     DA         Replaced the native stream by Clouds3D8C2000Static
## -----
↪ -----

"""
Ver. 1.1.0 (2024-02-06)

This module demonstrates the principles of stream processing with MLPro. To this regard,
↪ a stream of
a stream provider is combined with a stream workflow to a stream scenario. The workflow_
↪ consists of
a custom task only. The stream scenario is used to process some instances.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

```

(continues on next page)

(continued from previous page)

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with visualization.

"""

```
from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
```

```
## -----
↳ -----
```

```
## -----
↳ -----
```

```
class MyTask (StreamTask):
```

```
    """
```

```
    Demo implementation of a stream task with custom method _run().
```

```
    """
```

```
    # needed for proper logging (see class mlpro.bf.various.Log)
```

```
    C_NAME      = 'Custom'
```

```
## -----
↳ -----
```

```
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass
```

```
## -----
↳ -----
```

```
## -----
↳ -----
```

```
class MyScenario (StreamScenario):
```

```
    """
```

```
    Example of a custom stream scenario including a stream and a stream workflow. See
```

```
↳ class
```

```
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
```

```
    """
```

```
    C_NAME      = 'Demo'
```

```
## -----
↳ -----
```

```
    def _setup(self, p_mode, p_visualize: bool, p_logging):
```

```
        # 1 Import a stream from OpenML
```

(continues on next page)

(continued from previous page)

```

provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
stream = provider_mlpro.get_stream('Clouds3D8C2000Static', p_logging=p_logging)

# 2 Set up a stream workflow
workflow = StreamWorkflow( p_name='wf1',
                           p_range_max=Task.C_RANGE_NONE,
                           p_visualize=p_visualize,
                           p_logging=logging )

# 2.1 Set up and add an own custom task
task = MyTask( p_name='t1', p_visualize=p_visualize, p_logging=logging )
workflow.add_task( p_task=task )

# 3 Return stream and workflow
return stream, workflow

# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    cycle_limit = 721
    logging      = Log.C_LOG_ALL
    visualize    = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                          p_cycle_limit=cycle_limit,
                          p_visualize=visualize,
                          p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot()
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')

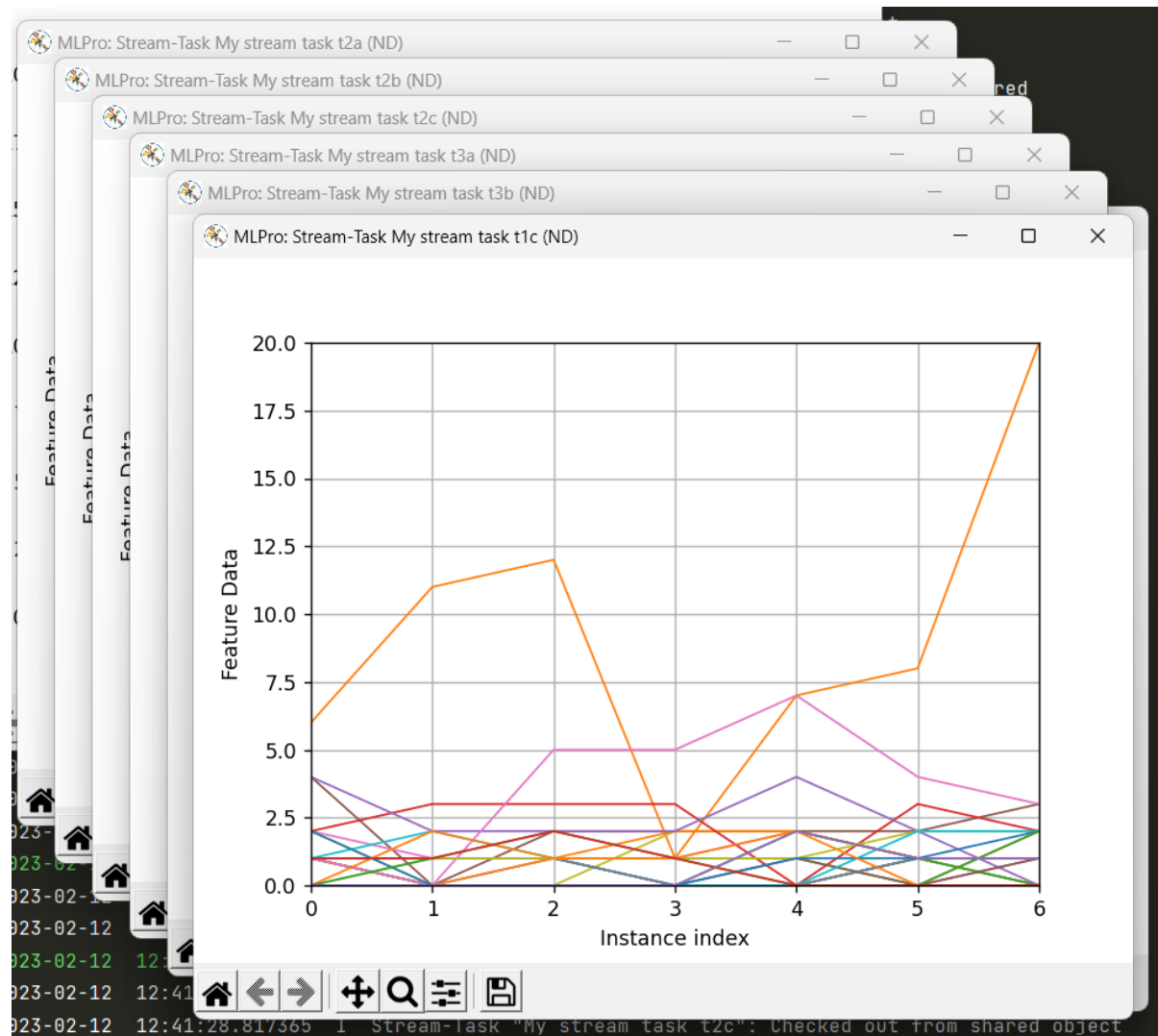
```

## Results



**Cross Reference**

- [API Reference: Streams](#)

**Howto BF-STREAMS-102: Tasks Workflows And Stream Scenarios****Executable code****Results****Cross Reference**

- [API Reference: Streams](#)

## Howto BF-STREAMS-110: Window

### Prerequisites

Please install the following packages to run this example properly:

- Numpy
- Matplotlib

### Executable code

```
## -----
↪-----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_110_stream_task_window.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-11-27  1.0.0     LSB      Creation
## -- 2022-12-14  1.1.0     DA       - Changed the stream provider from OpenML to MLPro
## --                                     - Added a custom task behind the window task
## -----
↪-----

"""
Ver. 1.0.1 (2022-12-14)

This module demonstrates the functionality of stream window task in MLPro.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with default visualization.
"""

from mlpro.bf.streams.streams import *
from mlpro.bf.streams.tasks import Window

## -----
↪-----
## -----
↪-----
class MyTask (StreamTask):
    """
```

(continues on next page)

(continued from previous page)

```

Demo implementation of a stream task with custom method _run().
"""

# needed for proper logging (see class mlpro.bf.various.Log)
C_NAME      = 'Custom'

## -----
↪ -----
def _run(self, p_inst_new: list, p_inst_del: list):
    pass

## -----
↪ -----
## -----
↪ -----
class MyStreamScenario(StreamScenario):

    C_NAME      = 'Demo Window'

    ## -----
    ↪ -----
    def _setup(self, p_mode, p_visualize:bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
        stream = provider_mlpro.get_stream('Rnd10Dx1000', p_mode=p_mode, p_logging=p_
↪ logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf-window',
                                   p_range_max=StreamWorkflow.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging)

        # 2.1 Set up and add a window task
        task_window = Window( p_buffer_size=30,
                              p_name = 't1',
                              p_delay = True,
                              p_visualize = p_visualize,
                              p_enable_statistics = True,
                              p_logging = p_logging)

        workflow.add_task(task_window)

        # 2.2 Set up and add an own custom task
        task_custom = MyTask( p_name='t2', p_visualize=p_visualize, p_logging=logging )
        workflow.add_task( p_task=task_custom, p_pred_tasks=[task_window] )

```

(continues on next page)

(continued from previous page)

```
# 3 Return stream and workflow
return stream, workflow

if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    cycle_limit = 100
    logging = Log.C_LOG_ALL
    visualize = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging = Log.C_LOG_NOTHING
    visualize = False

# 2 Instantiate the stream scenario
myscenario = MyStreamScenario(p_mode=Mode.C_MODE_REAL,
                              p_cycle_limit=cycle_limit,
                              p_visualize=visualize,
                              p_logging=logging)

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot()
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

## Results

## Cross Reference

- *API Reference: Streams*

## Howto BF-STREAMS-111: Rearranger (2D)

### Prerequisites

Please install the following packages to run this example properly:

- Matplotlib
- Tkinter

### Executable code

```
## -----
↪-----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_111_stream_task_rearranger_2d.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2022-10-27  0.0.0     DA         Creation
## -- 2022-12-14  1.0.0     DA         First implementation
## -----
↪-----

"""
Ver. 1.0.0 (2022-12-14)

This module demonstrates the principles of stream processing with MLPro. To this regard,↪
↪a stream of
a stream provider is combined with a stream workflow to a stream scenario. The workflow↪
↪consists of
a standard task Rearranger and a custom task. The stream scenario is used to process↪
↪some instances.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with visualization.

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.streams.tasks import Rearranger
```

(continues on next page)

(continued from previous page)

```

## -----
↪ -----
## -----
↪ -----
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'Custom'

## -----
↪ -----
def _run(self, p_inst_new: list, p_inst_del: list):
    pass

## -----
↪ -----
## -----
↪ -----
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See
↪ class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'Demo Rearranger'

## -----
↪ -----
def _setup(self, p_mode, p_visualize: bool, p_logging):

    # 1 Import a native stream from MLPro
    provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
    stream = provider_mlpro.get_stream('Rnd10Dx1000', p_mode=p_mode, p_logging=p_
↪ logging)

    # 2 Set up a stream workflow
    workflow = StreamWorkflow( p_name='wf1',
                               p_range_max=Task.C_RANGE_NONE,
                               p_visualize=p_visualize,
                               p_logging=logging )

    # 2.1 Set up and add a rearranger task to reduce the feature and label space
    features      = stream.get_feature_space().get_dims()
    features_new = [ ( 'F', features[1:3] ) ]

    task_rearranger = Rearranger( p_name='t1',

```

(continues on next page)

(continued from previous page)

```

        p_range_max=Task.C_RANGE_THREAD,
        p_visualize=p_visualize,
        p_logging=p_logging,
        p_features_new=features_new )

    workflow.add_task( p_task=task_rearranger )

    # 2.2 Set up and add an own custom task
    task_custom = MyTask( p_name='t2', p_visualize=p_visualize, p_logging=logging )
    workflow.add_task( p_task=task_custom, p_pred_tasks=[task_rearranger] )

    # 3 Return stream and workflow
    return stream, workflow

# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    cycle_limit = 200
    logging      = Log.C_LOG_ALL
    visualize    = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                        p_cycle_limit=cycle_limit,
                        p_visualize=visualize,
                        p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                       p_step_rate = 2 ) )

    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

## Results

**Cross Reference**

- *API Reference: Streams*

**Howto BF-STREAMS-112: Rearranger (3D)****Prerequisites**

Please install the following packages to run this example properly:

- Matplotlib
- Tkinter

**Executable code**

```
## -----
↪ -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_112_stream_task_rearranger_3d.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-27  0.0.0     DA       Creation
## -- 2022-12-14  1.0.0     DA       First implementation
## -- 2023-02-07  1.0.1     SY       Refactoring module name
## -----
↪ -----
"""
Ver. 1.0.1 (2023-02-07)

This module demonstrates the principles of stream processing with MLPro. To this regard,↪
↪ a stream of
a stream provider is combined with a stream workflow to a stream scenario. The workflow↪
↪ consists of
a standard task Rearranger and a custom task. The stream scenario is used to process↪
↪ some instances.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with visualization.

"""

from mlpro.bf.streams import *
```

(continues on next page)



(continued from previous page)

```

from mlpro.bf.streams.streams import *
from mlpro.bf.streams.tasks import Rearranger

## -----
↪ -----
## -----
↪ -----
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'Custom'

## -----
↪ -----
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass

## -----
↪ -----
## -----
↪ -----
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See ↪
    ↪ class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'Demo Rearranger'

## -----
↪ -----
    def _setup(self, p_mode, p_visualize: bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
        stream = provider_mlpro.get_stream('Rnd10Dx1000', p_mode=p_mode, p_logging=p_
↪ logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=Task.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=p_logging )

```

(continues on next page)

(continued from previous page)

```

# 2.1 Set up and add a rearranger task to reduce the feature and label space
features      = stream.get_feature_space().get_dims()
features_new = [ ( 'F', features[1:4] ) ]

task_rearranger = Rearranger( p_name='t1',
                              p_range_max=Task.C_RANGE_THREAD,
                              p_visualize=p_visualize,
                              p_logging=p_logging,
                              p_features_new=features_new )

workflow.add_task( p_task=task_rearranger )

# 2.2 Set up and add an own custom task
task_custom = MyTask( p_name='t2', p_visualize=p_visualize, p_logging=p_logging )
workflow.add_task( p_task=task_custom, p_pred_tasks=[task_rearranger] )

# 3 Return stream and workflow
return stream, workflow

# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    cycle_limit = 200
    logging      = Log.C_LOG_ALL
    visualize    = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                          p_cycle_limit=cycle_limit,
                          p_visualize=visualize,
                          p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                         p_step_rate = 2 ) )
    input('Press ENTER to start stream processing...')

```

(continues on next page)

(continued from previous page)

```
myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

## Results

## Cross Reference

- *API Reference: Streams*

## Howto BF-STREAMS-113: Rearranger (nD)

## Prerequisites

Please install the following packages to run this example properly:

- Matplotlib
- Tkinter

## Executable code

```
## -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_113_stream_task_rearranger_nd.py
## -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2022-10-27  0.0.0     DA         Creation
## -- 2022-12-14  1.0.0     DA         First implementation
## -- 2023-02-07  1.0.1     SY         Refactoring module name
## -----
"""
Ver. 1.0.1 (2023-02-07)

This module demonstrates the principles of stream processing with MLPro. To this regard,
↳ a stream of
a stream provider is combined with a stream workflow to a stream scenario. The workflow
↳ consists of
a standard task Rearranger and a custom task. The stream scenario is used to process
↳ some instances.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.
```

(continues on next page)

(continued from previous page)

- 3) How to set up a stream scenario based on a stream and a processing stream workflow.
- 4) How to run a stream scenario dark or with visualization.

```

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.streams.tasks import Rearranger

## -----
↪ -----
## -----
↪ -----
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'Custom'

## -----
↪ -----
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass

## -----
↪ -----
## -----
↪ -----
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See
↪ class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'Demo Rearranger'

## -----
↪ -----
    def _setup(self, p_mode, p_visualize: bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_logging=p_logging)

```

(continues on next page)

(continued from previous page)

```

stream = provider_mlpro.get_stream('Rnd10Dx1000', p_mode=p_mode, p_logging=p_
↳ logging)

# 2 Set up a stream workflow
workflow = StreamWorkflow( p_name='wf1',
                           p_range_max=Task.C_RANGE_NONE,
                           p_visualize=p_visualize,
                           p_logging=logging )

# 2.1 Set up and add a rearranger task to reduce the feature and label space
features = stream.get_feature_space().get_dims()
labels    = stream.get_label_space().get_dims()

features_new = [ ( 'F', [ features[1] ] ),
                  ( 'L', [ labels[1] ] ),
                  ( 'F', features[5:8] ) ]
labels_new   = [ ( 'L', [ labels[0] ] ),
                  ( 'F', features[4:6] ) ]

task_rearranger = Rearranger( p_name='t1',
                              p_range_max=Task.C_RANGE_THREAD,
                              p_visualize=p_visualize,
                              p_logging=p_logging,
                              p_features_new=features_new,
                              p_labels_new=labels_new )

workflow.add_task( p_task=task_rearranger )

# 2.2 Set up and add an own custom task
task_custom = MyTask( p_name='t2', p_visualize=p_visualize, p_logging=logging )
workflow.add_task( p_task=task_custom, p_pred_tasks=[task_rearranger] )

# 3 Return stream and workflow
return stream, workflow

# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    cycle_limit = 200
    logging      = Log.C_LOG_ALL
    visualize    = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False

```

(continues on next page)

(continued from previous page)

```

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                        p_cycle_limit=cycle_limit,
                        p_visualize=visualize,
                        p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                    p_step_rate = 2 ) )
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')

```

## Results

## Cross Reference

- *API Reference: Streams*

## Howto BF-STREAMS-114: Deriver

## Prerequisites

Please install the following packages to run this example properly:

- Matplotlib
- Tkinter

## Executable code

```

## -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_114_stream_task_deriver.py
## -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2023-02-02  0.0.0     SY          Creation
## -- 2023-02-05  1.0.0     SY          First version release
## -- 2023-02-07  1.1.0     SY          Change the dataset to doublespiral2d
## -- 2023-02-12  1.2.0     DA          New plot parameter p_view_autoselect
## -----

```

(continues on next page)

(continued from previous page)

```

"""
Ver. 1.2.0 (2023-02-12)

This module demonstrates the principles of stream processing with MLPro. To this regard,
↳ a stream of
a stream provider is combined with a stream workflow to a stream scenario. The workflow
↳ consists of
a standard task Deriver and a custom task. The stream scenario is used to process some
↳ instances.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to add a task Deriver and how to extend the features.

5) How to run a stream scenario dark or with visualization.

"""

from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.streams.tasks import Rearranger, Deriver

## -----
↳ -----
## -----
↳ -----
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'Custom'

## -----
↳ -----
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass

## -----

```

(continues on next page)

(continued from previous page)

```

↪ -----
## -----
↪ -----
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See
↪ class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'Demo Deriver'

## -----
↪ -----
    def _setup(self, p_mode, p_visualize: bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
        stream = provider_mlpro.get_stream('DoubleSpiral2D', p_mode=p_mode, p_logging=p_
↪ logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=Task.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 2.1 Set up and add a rearranger task to reduce the feature and label space
        features = stream.get_feature_space().get_dims()
        features_new = [ ( 'F', features[0:1] ) ]

        task_rearranger = Rearranger( p_name='t1',
                                       p_range_max=Task.C_RANGE_THREAD,
                                       p_visualize=p_visualize,
                                       p_logging=p_logging,
                                       p_features_new=features_new )

        workflow.add_task( p_task=task_rearranger )

        # 2.2 Set up and add a deriver task to extend the feature and label space (1st
↪ derivative)
        features = task_rearranger._feature_space.get_dims()
        derived_feature = features[0]

        task_deriver_1 = Deriver( p_name='t2',
                                   p_range_max=Task.C_RANGE_THREAD,
                                   p_visualize=p_visualize,
                                   p_logging=p_logging,
                                   p_features=features,
                                   p_label=None,
                                   p_derived_feature=derived_feature,
                                   p_derived_label=None,

```

(continues on next page)



(continued from previous page)

```

                                p_order_derivative=1 )

    workflow.add_task( p_task=task_deriver_1, p_pred_tasks=[task_rearranger] )

    # 2.3 Set up and add a deriver task to extend the feature and label space (2nd_
    ↪derivative)
    features = task_deriver_1._feature_space.get_dims()
    derived_feature = features[0]

    task_deriver_2 = Deriver( p_name='t3',
                              p_range_max=Task.C_RANGE_THREAD,
                              p_visualize=p_visualize,
                              p_logging=p_logging,
                              p_features=features,
                              p_label=None,
                              p_derived_feature=derived_feature,
                              p_derived_label=None,
                              p_order_derivative=2 )

    workflow.add_task( p_task=task_deriver_2, p_pred_tasks=[task_rearranger, task_
    ↪deriver_1] )

    # 3 Return stream and workflow
    return stream, workflow

# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    cycle_limit = 200
    logging      = Log.C_LOG_ALL
    visualize    = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging      = Log.C_LOG_NOTHING
    visualize    = False

# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                          p_cycle_limit=cycle_limit,
                          p_visualize=visualize,
                          p_logging=logging )

# 3 Reset and run own stream scenario
myscenario.reset()

```

(continues on next page)

(continued from previous page)

```

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                       p_view_autoselect = False,
                                                       p_step_rate = 2 ) )

    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')

```

## Results

## Cross Reference

- *API Reference: Deriver*

## Physics

### Howto BF-PHYSICS-001: Transfer Functions

## Prerequisites

Please install following packages to run this how to

- Matplotlib

## Executable code

```

## -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_physics_001_set_up_transfer_functions.py
## -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2022-08-24  0.0.0     SY/ML      Creation
## -- 2022-11-22  0.0.1     SY         Shift from mlpro repo to mlpro-mpps repo
## -- 2022-11-22  1.0.0     SY/ML      Release of first version
## -- 2023-01-14  1.0.1     SY         Shift from mlpro-at_basis.bf to mlpro.bf
## -- 2023-01-15  1.0.2     SY         Package renaming
## -- 2023-01-16  1.0.3     SY         Update due to __call__
## -- 2023-01-24  1.0.4     SY         Quality Assurance on TransferFunction
## -- 2023-02-04  1.0.5     SY         Shift UnitConverter from bf.systems to bf.physics
## -- 2023-03-14  1.0.6     SY         Recatoring
## -----
"""
Ver. 1.0.6 (2023-03-14)

```

(continues on next page)

(continued from previous page)

This module provides an example of using the transfer function method in MLPro for both `↪` default and custom implementation.

You will learn:

- 1) How to use the default type of transfer function (linear function)
- 2) How to set up your own function

"""

```
from mlpro.bf.math import *
from mlpro.bf.physics import *
import math
```

```
if __name__ == "__main__":
    p_logging = Log.C_LOG_ALL
    p_print = True
    p_visualize = True
else:
    p_logging = Log.C_LOG_NOTHING
    p_print = False
    p_visualize = False
```

# 1. Using default type

# 1.1. Initialize a given default transfer function

```
myTF_linear = TransferFunction(p_name='Linear_TF',
                               p_type=TransferFunction.C_TRF_FUNC_LINEAR,
                               p_logging=p_logging,
                               p_dt=0.01,
                               m=5,
                               b=2)
```

# 1.2. Call the defined transfer function

# 1.2.1. For a specific point

```
p_input = 10
output = myTF_linear(p_input)
if p_print:
    print(output)
```

# 1.2.2. Within a specific range

```
p_range = 5
output = myTF_linear(p_input, p_range)
if p_print:
```

(continues on next page)

(continued from previous page)

```

    print(output)

# 1.3. Plot the graph
if p_visualize:
    myTF_linear.plot(p_input, p_input+p_range)

# 2. Using own function

# 2.1. Set up your custom function class
class MyTransferFunction(TransferFunction):

    # 2.1.1. Set up which parameters required for your transfer function
    def _set_function_parameters(self, p_args) -> bool:
        """
         $y(t) = A \cos(w * t - \phi)$ 
        """
        if self.get_type() == self.C_TRF_FUNC_CUSTOM:
            try:
                self.A = p_args['A']
                self.w = p_args['w']
                self.phi = p_args['phi']
            except:
                raise NotImplementedError('One/More parameters for this function is_
↳missing.')
            return True

    # 2.1.2. Set up the mathematical calculation of your transfer function
    def _custom_function(self, p_input, p_range=None):
        """
         $y(t) = A \cos(w * t - \phi)$ 
        """
        if p_range is None:
            return self.A * math.cos(self.w * p_input - self.phi)
        else:
            points = int(p_range/self.dt)
            output = 0
            for x in range(points+1):
                current_input = p_input + x * self.dt
                output += self.A * math.cos(self.w * current_input - self.phi)
            return output

# 2.2. Initialize the transfer function
myFunction = MyTransferFunction(p_name='DGL_solution',
                                p_type=TransferFunction.C_TRF_FUNC_CUSTOM,
                                p_logging=p_logging,
                                p_dt=0.05,
                                A = 3.5,           # Current
                                w = 314.15,        # angular velocity
                                phi = -120)        # angle offset

```

(continues on next page)

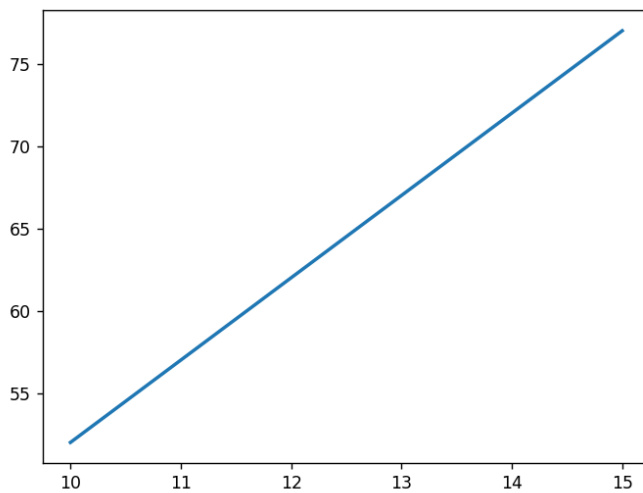
(continued from previous page)

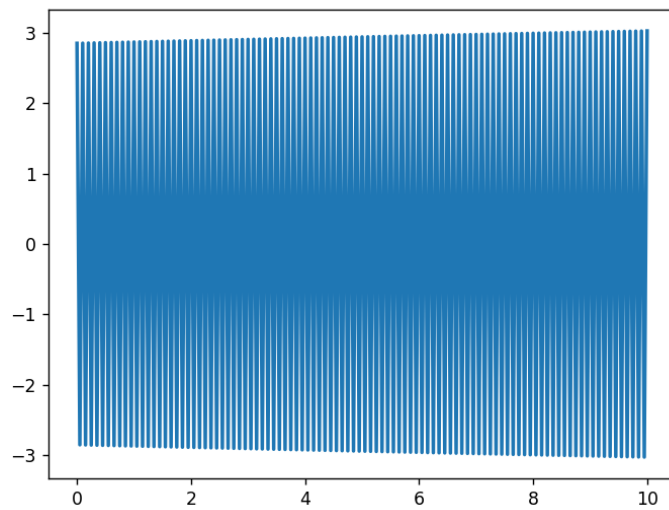
```
# 2.3. Call the defined transfer function
# 2.3.1. For a specific point
p_input = 0
output = myFunction(p_input)
if p_print:
    print(output)

# 2.3.2. Within a specific range
p_range = 10
output = myFunction(p_input, p_range)
if p_print:
    print(output)

# 2.4. Plot the graph
if p_visualize:
    myFunction.plot(p_input, p_input+p_range)
```

## Results





### Cross Reference

- API Reference: Physics

### Howto BF-PHYSICS-002: Unit Converter

#### Executable code

```
## -----
↪ -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_physics_002_unit_converter.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-01-15  0.0.0     SY       Creation
## -- 2023-01-15  1.0.0     SY       Release of first version
## -- 2023-01-16  1.0.1     SY       Renaming and debugging
## -- 2023-02-04  1.0.2     SY       Shift UnitConverter from bf.math to bf.physics
## -- 2023-03-14  1.0.3     SY       Recatoring
## -----
↪ -----

"""
Ver. 1.0.3 (2023-03-14)

This module provides an example of using the unit converter in MLPro.

You will learn:

1) How to use the the unit converter
```

(continues on next page)

(continued from previous page)

```

"""

from mlpro.bf.physics.unitconverter import UnitConverter
from mlpro.bf.various import Log

if __name__ == "__main__":
    p_logging = Log.C_LOG_ALL
    p_print = True
else:
    p_logging = Log.C_LOG_NOTHING
    p_print = False

# 1 Initialize unit converters
conv_length = UnitConverter(p_name='conv_length',
                             p_type=UnitConverter.C_UNIT_CONV_LENGTH,
                             p_logging=p_logging,
                             p_unit_in='m',
                             p_unit_out='km')

conv_pressure = UnitConverter(p_name='conv_pressure',
                               p_type=UnitConverter.C_UNIT_CONV_PRESSURE,
                               p_logging=p_logging,
                               p_unit_in='bar',
                               p_unit_out='Pa')

conv_current = UnitConverter(p_name='conv_current',
                              p_type=UnitConverter.C_UNIT_CONV_CURRENT,
                              p_logging=p_logging,
                              p_unit_in='mA',
                              p_unit_out='A')

conv_force = UnitConverter(p_name='conv_force',
                            p_type=UnitConverter.C_UNIT_CONV_FORCE,
                            p_logging=p_logging,
                            p_unit_in='N',
                            p_unit_out='J/cm')

conv_power = UnitConverter(p_name='conv_power',
                            p_type=UnitConverter.C_UNIT_CONV_POWER,
                            p_logging=p_logging,
                            p_unit_in='W',
                            p_unit_out='kW')

conv_mass = UnitConverter(p_name='conv_mass',
                           p_type=UnitConverter.C_UNIT_CONV_MASS,
                           p_logging=p_logging,
                           p_unit_in='kg',
                           p_unit_out='lb')

```

(continues on next page)

(continued from previous page)

```

conv_time = UnitConverter(p_name='conv_time',
                          p_type=UnitConverter.C_UNIT_CONV_TIME,
                          p_logging=p_logging,
                          p_unit_in='hr',
                          p_unit_out='s')

conv_temperature = UnitConverter(p_name='conv_temperature',
                                 p_type=UnitConverter.C_UNIT_CONV_TEMPERATURE,
                                 p_logging=p_logging,
                                 p_unit_in='K',
                                 p_unit_out='F')

# 2. Call the defined unit converters
conv_set = [conv_length,
            conv_pressure,
            conv_current,
            conv_force,
            conv_power,
            conv_mass,
            conv_time,
            conv_temperature]

p_input = 10

for conv in conv_set:
    output = conv(p_input)
    if p_print:
        print('We convert %.1f%s to %.2f%s'%(p_input, conv._unit_in, output, conv._unit_
        ↪out))

```

## Results

```

2023-02-10 17:54:42.123738 I TransferFunction "Linear_TF": Instantiated
52
32314.5
2023-02-10 17:55:14.475198 I TransferFunction "DGL_solution": Instantiated
2.8496333968429663
2.937498276638952

```

## Cross Reference

- API Reference: Physics



## State-based Systems

### Howto BF-SYSTEMS-001: Demonstrating Native Systems

#### Executable code

```
## -----
↳ -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_systems_001.py
## -----
↳ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2023-04-19  0.0.0     LSB        Creation
## -- 2023-04-19  1.0.0     LSB        Release
## -- 2023-05-03  1.0.1     LSB        - Dark Mode
##                                     - Visuaization
## -----
↳ -----

"""
This module illustrates the use of DemoScenario to demonstrate systems. This example
↳ runs all the
native systems in MLPro for 10 cycles.

You will learn:

1. Setting up demo scenario
2. Running a system in MLPro

ver. 1.0.1 (2023-05-03)

"""

from mlpro.bf.systems import *
from mlpro.bf.systems.pool import DoublePendulumSystemS4, DoublePendulumSystemS7

# Checking for dark mode:
if __name__ == '__main__':
    logging = Log.C_LOG_ALL
    visualize = True
    cycle_limit = 10
else:
    logging = Log.C_LOG_NOTHING
    visualize = False
    cycle_limit = 2

systems = [DoublePendulumSystemS4(p_logging=logging, p_visualize=visualize),
```

(continues on next page)

(continued from previous page)

```

↪DoublePendulumSystemS7(p_logging=logging, p_visualize=visualize)]

for system in systems:

    scenario = DemoScenario(p_system=system,
                            p_mode = Mode.C_MODE_SIM,
                            p_action_pattern = DemoScenario.C_ACTION_RANDOM,
                            p_cycle_limit=cycle_limit,
                            p_visualize=visualize,
                            p_logging=logging)

    scenario.run()

```

## Results

```

2023-05-03 10:50:48.419856 I System "DoublePendulumSystemS4": Instantiated
2023-05-03 10:50:48.419856 I System "DoublePendulumSystemS4": Reset
2023-05-03 10:50:48.420853 I System "DoublePendulumSystemS7": Instantiated
2023-05-03 10:50:48.420853 I System "DoublePendulumSystemS7": Reset
2023-05-03 10:50:48.420853 I Scenario Base "????": Instantiated
2023-05-03 10:50:48.420853 I Scenario Base "????": Process time 0:00:00 : Scenario_
↪reset with seed 1
2023-05-03 10:50:48.421853 I System "DoublePendulumSystemS4": Reset
2023-05-03 10:50:48.421853 S Scenario Base "????": Process time 0:00:00 : Start of_
↪processing
2023-05-03 10:50:48.421853 S Scenario Base "????": Process time 0:00:00 : Start of_
↪cycle 0
2023-05-03 10:50:48.421853 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.421853 I System "DoublePendulumSystemS4": Start processing action
2023-05-03 10:50:48.421853 I System "DoublePendulumSystemS4": Actions of agent 0 = [-
↪0.18259651632236285]
2023-05-03 10:50:48.422850 I System "DoublePendulumSystemS4": Assessment for success..
↪.
2023-05-03 10:50:48.423849 I System "DoublePendulumSystemS4": Assessment for_
↪breakdown...
2023-05-03 10:50:48.423849 I System "DoublePendulumSystemS4": Action processing_
↪finished successfully
2023-05-03 10:50:48.423849 I System "DoublePendulumSystemS4": Assessment for_
↪breakdown...
2023-05-03 10:50:48.423849 S Scenario Base "????": Process time 0:00:00 : End of_
↪cycle 0
2023-05-03 10:50:48.424853 S Scenario Base "????": Process time 0:00:00.040000 :_
↪Start of cycle 1
2023-05-03 10:50:48.424853 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.424853 I System "DoublePendulumSystemS4": Start processing action
2023-05-03 10:50:48.424853 I System "DoublePendulumSystemS4": Actions of agent 0 = [-
↪2.020357408450476]
2023-05-03 10:50:48.426856 I System "DoublePendulumSystemS4": Assessment for success..
↪.
2023-05-03 10:50:48.427853 I System "DoublePendulumSystemS4": Assessment for_
↪breakdown...

```

(continues on next page)

(continued from previous page)

```

2023-05-03 10:50:48.427853 I System "DoublePendulumSystemS4": Action processing_
↳ finished successfully
2023-05-03 10:50:48.427853 I System "DoublePendulumSystemS4": Assessment for_
↳ breakdown...
2023-05-03 10:50:48.428854 S Scenario Base "????": Process time 0:00:00.040000 : End_
↳ of cycle 1
2023-05-03 10:50:48.428854 S Scenario Base "????": Process time 0:00:00.080000 :_
↳ Start of cycle 2
2023-05-03 10:50:48.428854 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.428854 I System "DoublePendulumSystemS4": Start processing action
2023-05-03 10:50:48.429852 I System "DoublePendulumSystemS4": Actions of agent 0 = [6.
↳ 063718908910516]
2023-05-03 10:50:48.431851 I System "DoublePendulumSystemS4": Assessment for success..
↳ .
2023-05-03 10:50:48.431851 I System "DoublePendulumSystemS4": Assessment for_
↳ breakdown...
2023-05-03 10:50:48.431851 I System "DoublePendulumSystemS4": Action processing_
↳ finished successfully
2023-05-03 10:50:48.432867 I System "DoublePendulumSystemS4": Assessment for_
↳ breakdown...
2023-05-03 10:50:48.432867 S Scenario Base "????": Process time 0:00:00.080000 : End_
↳ of cycle 2
2023-05-03 10:50:48.432867 S Scenario Base "????": Process time 0:00:00.120000 :_
↳ Start of cycle 3
2023-05-03 10:50:48.433854 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.433854 I System "DoublePendulumSystemS4": Start processing action
2023-05-03 10:50:48.433854 I System "DoublePendulumSystemS4": Actions of agent 0 =_
↳ [11.548934045420527]
2023-05-03 10:50:48.435851 I System "DoublePendulumSystemS4": Assessment for success..
↳ .
2023-05-03 10:50:48.435851 I System "DoublePendulumSystemS4": Assessment for_
↳ breakdown...
2023-05-03 10:50:48.435851 I System "DoublePendulumSystemS4": Action processing_
↳ finished successfully
2023-05-03 10:50:48.436852 I System "DoublePendulumSystemS4": Assessment for_
↳ breakdown...
2023-05-03 10:50:48.436852 S Scenario Base "????": Process time 0:00:00.120000 : End_
↳ of cycle 3
2023-05-03 10:50:48.438163 S Scenario Base "????": Process time 0:00:00.160000 :_
↳ Start of cycle 4
2023-05-03 10:50:48.438163 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.438862 I System "DoublePendulumSystemS4": Start processing action
2023-05-03 10:50:48.438862 I System "DoublePendulumSystemS4": Actions of agent 0 = [-
↳ 16.245616529030606]
2023-05-03 10:50:48.440850 I System "DoublePendulumSystemS4": Assessment for success..
↳ .
2023-05-03 10:50:48.440850 I System "DoublePendulumSystemS4": Assessment for_
↳ breakdown...
2023-05-03 10:50:48.440850 I System "DoublePendulumSystemS4": Action processing_
↳ finished successfully
2023-05-03 10:50:48.441852 I System "DoublePendulumSystemS4": Assessment for_
↳ breakdown...

```

(continues on next page)

(continued from previous page)

```

2023-05-03 10:50:48.441852 S Scenario Base "????": Process time 0:00:00.160000 : End_
↳of cycle 4
2023-05-03 10:50:48.441852 S Scenario Base "????": Process time 0:00:00.200000 :_
↳Start of cycle 5
2023-05-03 10:50:48.441852 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.442852 I System "DoublePendulumSystemS4": Start processing action
2023-05-03 10:50:48.442852 I System "DoublePendulumSystemS4": Actions of agent 0 = [-
↳18.866100939119747]
2023-05-03 10:50:48.445850 I System "DoublePendulumSystemS4": Assessment for success..
↳.
2023-05-03 10:50:48.445850 I System "DoublePendulumSystemS4": Assessment for_
↳breakdown...
2023-05-03 10:50:48.446849 I System "DoublePendulumSystemS4": Action processing_
↳finished successfully
2023-05-03 10:50:48.446849 I System "DoublePendulumSystemS4": Assessment for_
↳breakdown...
2023-05-03 10:50:48.446849 S Scenario Base "????": Process time 0:00:00.200000 : End_
↳of cycle 5
2023-05-03 10:50:48.446849 S Scenario Base "????": Process time 0:00:00.240000 :_
↳Start of cycle 6
2023-05-03 10:50:48.446849 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.446849 I System "DoublePendulumSystemS4": Start processing action
2023-05-03 10:50:48.448365 I System "DoublePendulumSystemS4": Actions of agent 0 =_
↳[13.430604156794786]
2023-05-03 10:50:48.450378 I System "DoublePendulumSystemS4": Assessment for success..
↳.
2023-05-03 10:50:48.450378 I System "DoublePendulumSystemS4": Assessment for_
↳breakdown...
2023-05-03 10:50:48.450378 I System "DoublePendulumSystemS4": Action processing_
↳finished successfully
2023-05-03 10:50:48.451379 I System "DoublePendulumSystemS4": Assessment for_
↳breakdown...
2023-05-03 10:50:48.451379 S Scenario Base "????": Process time 0:00:00.240000 : End_
↳of cycle 6
2023-05-03 10:50:48.452385 S Scenario Base "????": Process time 0:00:00.280000 :_
↳Start of cycle 7
2023-05-03 10:50:48.454383 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.454383 I System "DoublePendulumSystemS4": Start processing action
2023-05-03 10:50:48.454383 I System "DoublePendulumSystemS4": Actions of agent 0 = [-
↳2.689317283797866]
2023-05-03 10:50:48.457380 I System "DoublePendulumSystemS4": Assessment for success..
↳.
2023-05-03 10:50:48.458390 I System "DoublePendulumSystemS4": Assessment for_
↳breakdown...
2023-05-03 10:50:48.458390 I System "DoublePendulumSystemS4": Action processing_
↳finished successfully
2023-05-03 10:50:48.459386 I System "DoublePendulumSystemS4": Assessment for_
↳breakdown...
2023-05-03 10:50:48.459386 S Scenario Base "????": Process time 0:00:00.280000 : End_
↳of cycle 7
2023-05-03 10:50:48.460387 S Scenario Base "????": Process time 0:00:00.320000 :_
↳Start of cycle 8

```

(continues on next page)

(continued from previous page)

```

2023-05-03 10:50:48.460387 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.461382 I System "DoublePendulumSystemS4": Start processing action
2023-05-03 10:50:48.462412 I System "DoublePendulumSystemS4": Actions of agent 0 = [
↳ [10.491203298317679]
2023-05-03 10:50:48.465387 I System "DoublePendulumSystemS4": Assessment for success..
↳ .
2023-05-03 10:50:48.466380 I System "DoublePendulumSystemS4": Assessment for
↳ breakdown...
2023-05-03 10:50:48.466380 I System "DoublePendulumSystemS4": Action processing
↳ finished successfully
2023-05-03 10:50:48.466380 I System "DoublePendulumSystemS4": Assessment for
↳ breakdown...
2023-05-03 10:50:48.467403 S Scenario Base "????": Process time 0:00:00.320000 : End
↳ of cycle 8
2023-05-03 10:50:48.467403 S Scenario Base "????": Process time 0:00:00.360000 :
↳ Start of cycle 9
2023-05-03 10:50:48.467403 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.467403 I System "DoublePendulumSystemS4": Start processing action
2023-05-03 10:50:48.468958 I System "DoublePendulumSystemS4": Actions of agent 0 = [-
↳ 19.915757865955573]
2023-05-03 10:50:48.471972 I System "DoublePendulumSystemS4": Assessment for success..
↳ .
2023-05-03 10:50:48.471972 I System "DoublePendulumSystemS4": Assessment for
↳ breakdown...
2023-05-03 10:50:48.471972 I System "DoublePendulumSystemS4": Action processing
↳ finished successfully
2023-05-03 10:50:48.473972 I System "DoublePendulumSystemS4": Assessment for
↳ breakdown...
2023-05-03 10:50:48.473972 S Scenario Base "????": Process time 0:00:00.360000 : End
↳ of cycle 9
2023-05-03 10:50:48.474970 S Scenario Base "????": Process time 0:00:00.400000 : End
↳ of processing
2023-05-03 10:50:48.474970 I Scenario Base "????": Instantiated
2023-05-03 10:50:48.475971 I Scenario Base "????": Process time 0:00:00 : Scenario
↳ reset with seed 1
2023-05-03 10:50:48.475971 I System "DoublePendulumSystemS7": Reset
2023-05-03 10:50:48.475971 S Scenario Base "????": Process time 0:00:00 : Start of
↳ processing
2023-05-03 10:50:48.476972 S Scenario Base "????": Process time 0:00:00 : Start of
↳ cycle 0
2023-05-03 10:50:48.476972 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.476972 I System "DoublePendulumSystemS7": Start processing action
2023-05-03 10:50:48.476972 I System "DoublePendulumSystemS7": Actions of agent 0 = [-
↳ 0.18259651632236285]
2023-05-03 10:50:48.478971 I System "DoublePendulumSystemS7": Assessment for success..
↳ .
2023-05-03 10:50:48.478971 I System "DoublePendulumSystemS7": Assessment for
↳ breakdown...
2023-05-03 10:50:48.478971 I System "DoublePendulumSystemS7": Action processing
↳ finished successfully
2023-05-03 10:50:48.480331 I System "DoublePendulumSystemS7": Assessment for
↳ breakdown...

```

(continues on next page)

(continued from previous page)

```

2023-05-03 10:50:48.481324 S Scenario Base "????": Process time 0:00:00 : End of
↳cycle 0
2023-05-03 10:50:48.481324 S Scenario Base "????": Process time 0:00:00.040000 :
↳Start of cycle 1
2023-05-03 10:50:48.482329 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.482329 I System "DoublePendulumSystemS7": Start processing action
2023-05-03 10:50:48.482329 I System "DoublePendulumSystemS7": Actions of agent 0 = [-
↳2.020357408450476]
2023-05-03 10:50:48.489836 I System "DoublePendulumSystemS7": Assessment for success..
↳.
2023-05-03 10:50:48.490859 I System "DoublePendulumSystemS7": Assessment for
↳breakdown...
2023-05-03 10:50:48.492870 I System "DoublePendulumSystemS7": Action processing
↳finished successfully
2023-05-03 10:50:48.493852 I System "DoublePendulumSystemS7": Assessment for
↳breakdown...
2023-05-03 10:50:48.493852 S Scenario Base "????": Process time 0:00:00.040000 : End
↳of cycle 1
2023-05-03 10:50:48.493852 S Scenario Base "????": Process time 0:00:00.080000 :
↳Start of cycle 2
2023-05-03 10:50:48.493852 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.494849 I System "DoublePendulumSystemS7": Start processing action
2023-05-03 10:50:48.494849 I System "DoublePendulumSystemS7": Actions of agent 0 = [6.
↳063718908910516]
2023-05-03 10:50:48.497848 I System "DoublePendulumSystemS7": Assessment for success..
↳.
2023-05-03 10:50:48.497848 I System "DoublePendulumSystemS7": Assessment for
↳breakdown...
2023-05-03 10:50:48.497848 I System "DoublePendulumSystemS7": Action processing
↳finished successfully
2023-05-03 10:50:48.497848 I System "DoublePendulumSystemS7": Assessment for
↳breakdown...
2023-05-03 10:50:48.498855 S Scenario Base "????": Process time 0:00:00.080000 : End
↳of cycle 2
2023-05-03 10:50:48.498855 S Scenario Base "????": Process time 0:00:00.120000 :
↳Start of cycle 3
2023-05-03 10:50:48.498855 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.498855 I System "DoublePendulumSystemS7": Start processing action
2023-05-03 10:50:48.499854 I System "DoublePendulumSystemS7": Actions of agent 0 =
↳[11.548934045420527]
2023-05-03 10:50:48.501851 I System "DoublePendulumSystemS7": Assessment for success..
↳.
2023-05-03 10:50:48.502853 I System "DoublePendulumSystemS7": Assessment for
↳breakdown...
2023-05-03 10:50:48.502853 I System "DoublePendulumSystemS7": Action processing
↳finished successfully
2023-05-03 10:50:48.502853 I System "DoublePendulumSystemS7": Assessment for
↳breakdown...
2023-05-03 10:50:48.502853 S Scenario Base "????": Process time 0:00:00.120000 : End
↳of cycle 3
2023-05-03 10:50:48.503855 S Scenario Base "????": Process time 0:00:00.160000 :
↳Start of cycle 4

```

(continues on next page)

(continued from previous page)

```

2023-05-03 10:50:48.503855 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.503855 I System "DoublePendulumSystemS7": Start processing action
2023-05-03 10:50:48.504868 I System "DoublePendulumSystemS7": Actions of agent 0 = [-
↳ 16.245616529030606]
2023-05-03 10:50:48.507846 I System "DoublePendulumSystemS7": Assessment for success..
↳ .
2023-05-03 10:50:48.507846 I System "DoublePendulumSystemS7": Assessment for↳
↳ breakdown...
2023-05-03 10:50:48.507846 I System "DoublePendulumSystemS7": Action processing↳
↳ finished successfully
2023-05-03 10:50:48.508850 I System "DoublePendulumSystemS7": Assessment for↳
↳ breakdown...
2023-05-03 10:50:48.508850 S Scenario Base "????": Process time 0:00:00.160000 : End↳
↳ of cycle 4
2023-05-03 10:50:48.508850 S Scenario Base "????": Process time 0:00:00.200000 :↳
↳ Start of cycle 5
2023-05-03 10:50:48.508850 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.508850 I System "DoublePendulumSystemS7": Start processing action
2023-05-03 10:50:48.510360 I System "DoublePendulumSystemS7": Actions of agent 0 = [-
↳ 18.866100939119747]
2023-05-03 10:50:48.512369 I System "DoublePendulumSystemS7": Assessment for success..
↳ .
2023-05-03 10:50:48.512369 I System "DoublePendulumSystemS7": Assessment for↳
↳ breakdown...
2023-05-03 10:50:48.513373 I System "DoublePendulumSystemS7": Action processing↳
↳ finished successfully
2023-05-03 10:50:48.513373 I System "DoublePendulumSystemS7": Assessment for↳
↳ breakdown...
2023-05-03 10:50:48.513373 S Scenario Base "????": Process time 0:00:00.200000 : End↳
↳ of cycle 5
2023-05-03 10:50:48.514376 S Scenario Base "????": Process time 0:00:00.240000 :↳
↳ Start of cycle 6
2023-05-03 10:50:48.514376 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.515382 I System "DoublePendulumSystemS7": Start processing action
2023-05-03 10:50:48.515382 I System "DoublePendulumSystemS7": Actions of agent 0 =↳
↳ [13.430604156794786]
2023-05-03 10:50:48.519427 I System "DoublePendulumSystemS7": Assessment for success..
↳ .
2023-05-03 10:50:48.520390 I System "DoublePendulumSystemS7": Assessment for↳
↳ breakdown...
2023-05-03 10:50:48.521408 I System "DoublePendulumSystemS7": Action processing↳
↳ finished successfully
2023-05-03 10:50:48.522373 I System "DoublePendulumSystemS7": Assessment for↳
↳ breakdown...
2023-05-03 10:50:48.522373 S Scenario Base "????": Process time 0:00:00.240000 : End↳
↳ of cycle 6
2023-05-03 10:50:48.522373 S Scenario Base "????": Process time 0:00:00.280000 :↳
↳ Start of cycle 7
2023-05-03 10:50:48.523373 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.523373 I System "DoublePendulumSystemS7": Start processing action
2023-05-03 10:50:48.524382 I System "DoublePendulumSystemS7": Actions of agent 0 = [-
↳ 2.689317283797866]

```

(continues on next page)



(continued from previous page)

```

2023-05-03 10:50:48.528376 I System "DoublePendulumSystemS7": Assessment for success..
↳.
2023-05-03 10:50:48.528376 I System "DoublePendulumSystemS7": Assessment for↳
↳breakdown...
2023-05-03 10:50:48.528376 I System "DoublePendulumSystemS7": Action processing↳
↳finished successfully
2023-05-03 10:50:48.529374 I System "DoublePendulumSystemS7": Assessment for↳
↳breakdown...
2023-05-03 10:50:48.529374 S Scenario Base "????": Process time 0:00:00.280000 : End↳
↳of cycle 7
2023-05-03 10:50:48.529374 S Scenario Base "????": Process time 0:00:00.320000 :↳
↳Start of cycle 8
2023-05-03 10:50:48.529374 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.529374 I System "DoublePendulumSystemS7": Start processing action
2023-05-03 10:50:48.530707 I System "DoublePendulumSystemS7": Actions of agent 0 =↳
↳[10.491203298317679]
2023-05-03 10:50:48.532373 I System "DoublePendulumSystemS7": Assessment for success..
↳.
2023-05-03 10:50:48.532373 I System "DoublePendulumSystemS7": Assessment for↳
↳breakdown...
2023-05-03 10:50:48.532373 I System "DoublePendulumSystemS7": Action processing↳
↳finished successfully
2023-05-03 10:50:48.533372 I System "DoublePendulumSystemS7": Assessment for↳
↳breakdown...
2023-05-03 10:50:48.533372 S Scenario Base "????": Process time 0:00:00.320000 : End↳
↳of cycle 8
2023-05-03 10:50:48.533372 S Scenario Base "????": Process time 0:00:00.360000 :↳
↳Start of cycle 9
2023-05-03 10:50:48.533372 I Scenario Base "????": Generating new action
2023-05-03 10:50:48.533372 I System "DoublePendulumSystemS7": Start processing action
2023-05-03 10:50:48.534372 I System "DoublePendulumSystemS7": Actions of agent 0 = [-
↳19.915757865955573]
2023-05-03 10:50:48.536372 I System "DoublePendulumSystemS7": Assessment for success..
↳.
2023-05-03 10:50:48.537369 I System "DoublePendulumSystemS7": Assessment for↳
↳breakdown...
2023-05-03 10:50:48.537369 I System "DoublePendulumSystemS7": Action processing↳
↳finished successfully
2023-05-03 10:50:48.537369 I System "DoublePendulumSystemS7": Assessment for↳
↳breakdown...
2023-05-03 10:50:48.537369 S Scenario Base "????": Process time 0:00:00.360000 : End↳
↳of cycle 9
2023-05-03 10:50:48.538374 S Scenario Base "????": Process time 0:00:00.400000 : End↳
↳of processing

```

## Cross Reference

- [API Reference: Systems](#)



## Howto BF-SYSTEMS-010: System, Controller, Actuator, Sensor

### Executable code

```
## -----
↪-----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_systems_010_systems_controllers_actuators_sensors.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2022-12-05  1.0.0     DA         Creation
## -- 2022-12-09  1.1.0     DA         Simplification
## -- 2023-04-19  1.1.1     LSB        Renamed the module
## -----
↪-----

"""
Ver. 1.1.1 (2023-04-19)

This module demonstrates the principles of using classes System, Controller, Actuator,
↪and Sensor. To
this regard we assume a custom system with two state and action components and a custom
↪controller
that represents the hardware pendant with two sensors and actuators.

You will learn:

1) How to set up an own state based system.

2) How to implement an own controller.

3) How to assign states to sensors and actions to actuators.

4) How to communicate with sensors and actuators using a controller.

"""

from mlpro.bf.various import Log
from mlpro.bf.systems import *
import random

class MyController (Controller):

    C_NAME      = 'Dummy'

    def _reset(self) -> bool:
        self.log(Log.C_LOG_TYPE_S, 'Pseudo-reset of the controller')
```

(continues on next page)

(continued from previous page)

```

    return True

def _get_sensor_value(self, p_id):
    """
    Pseudo-implementation for getting a sensor value.
    """
    self.log(Log.C_LOG_TYPE_S, 'Pseudo-import of a sensor value...')
    return random.random()

def _set_actuator_value(self, p_id, p_value) -> bool:
    self.log(Log.C_LOG_TYPE_S, 'Pseudo-export of an actuator value:', str(p_value))
    return True

class MySystem (System):

    C_NAME      = 'Dummy'

    @staticmethod
    def setup_spaces():

        # 1 State space
        state_space = ESpace()
        state_space.add_dim( p_dim = Dimension( p_name_short='State 1') )
        state_space.add_dim( p_dim = Dimension( p_name_short='State 2') )

        # 2 Action space
        action_space = ESpace()
        action_space.add_dim( p_dim = Dimension( p_name_short='Action 1') )
        action_space.add_dim( p_dim = Dimension( p_name_short='Action 2') )

        return state_space, action_space

    def _reset(self, p_seed=None) -> None:
        pass

# 0 Prepare Demo/Unit test mode
if __name__ == '__main__':
    logging      = Log.C_LOG_ALL
    latency      = timedelta(0,1,0)
else:
    logging      = Log.C_LOG_NOTHING
    latency      = timedelta(0,0,1000000)

```

(continues on next page)

(continued from previous page)

```

# 1 Instantiate own system in simulation mode
sys = MySystem( p_latency=latency, p_logging=logging )

# 2 Instantiate and configure own controller
con = MyController( p_id=0, p_name='2x2', p_logging=logging )

s1 = Sensor(p_name_short='Sensor 1')
s2 = Sensor(p_name_short='Sensor 2')
a1 = Actuator(p_name_short='Actuator 1')
a2 = Actuator(p_name_short='Actuator 2')

con.add_sensor( p_sensor=s1 )
con.add_sensor( p_sensor=s2 )
con.add_actuator( p_actuator=a1 )
con.add_actuator( p_actuator=a2 )

# 3 Add controller to system and assign sensors to states and actuators to actions
sys.add_controller( p_controller=con,
                   p_mapping=[ ( 'S', 'State 1', 'Sensor 1' ),
                               ( 'S', 'State 2', 'Sensor 2' ),
                               ( 'A', 'Action 1', 'Actuator 1' ),
                               ( 'A', 'Action 2', 'Actuator 2' ) ] )

# 4 Switch system to real mode
sys.set_mode( p_mode=Mode.C_MODE_REAL )

# 5 Reset system
sys.reset()

# 6 Process an action
sys.process_action( p_action= Action( p_agent_id=0,
                                     p_action_space=sys.get_action_space(),
                                     p_values=np.array([1,2]) ) )

```

## Results

```

2023-02-10 18:02:11.139147 I System "Dummy": Instantiated
2023-02-10 18:02:11.140156 I Controller "Dummy 2x2": Instantiated
2023-02-10 18:02:11.140156 I System "Dummy": Adding controller "Dummy 2x2"...
2023-02-10 18:02:11.140156 I System "Dummy": State component "State 1" assigned to ↵
↵sensor "Sensor 1"
2023-02-10 18:02:11.140156 I System "Dummy": State component "State 2" assigned to ↵
↵sensor "Sensor 2"

```

(continues on next page)

(continued from previous page)

```

2023-02-10 18:02:11.140156 I System "Dummy": Action component "Action 1" assigned to.
↳ actuator "Actuator 1"
2023-02-10 18:02:11.140156 I System "Dummy": Action component "Action 2" assigned to.
↳ actuator "Actuator 2"
2023-02-10 18:02:11.140156 I System "Dummy": Controller "Dummy 2x2" added successfully
2023-02-10 18:02:11.140156 I System "Dummy": Operation mode set to 1
2023-02-10 18:02:11.140156 I System "Dummy": Reset
2023-02-10 18:02:11.140156 I Controller "Dummy 2x2": Reset started
2023-02-10 18:02:11.140156 S Controller "Dummy 2x2": Pseudo-reset of the controller
2023-02-10 18:02:11.140156 I Controller "Dummy 2x2": Reset finished successfully
2023-02-10 18:02:11.141156 I System "Dummy": Start importing state...
2023-02-10 18:02:11.141156 I Controller "Dummy 2x2": Getting value of sensor "Sensor 1
↳ "...
2023-02-10 18:02:11.141156 S Controller "Dummy 2x2": Pseudo-import of a sensor value..
↳ .
2023-02-10 18:02:11.141156 I Controller "Dummy 2x2": Value of sensor "Sensor 1" = 0.
↳ 8862529182725326
2023-02-10 18:02:11.141156 I Controller "Dummy 2x2": Getting value of sensor "Sensor 2
↳ "...
2023-02-10 18:02:11.141156 S Controller "Dummy 2x2": Pseudo-import of a sensor value..
↳ .
2023-02-10 18:02:11.141156 I Controller "Dummy 2x2": Value of sensor "Sensor 2" = 0.
↳ 6398967672160553
2023-02-10 18:02:11.141156 I System "Dummy": Assessment for success...
2023-02-10 18:02:11.141156 I System "Dummy": Assessment for breakdown...
2023-02-10 18:02:11.141156 I System "Dummy": Start processing action
2023-02-10 18:02:11.141156 I System "Dummy": Actions of agent 0 = [1 2]
2023-02-10 18:02:11.142153 I System "Dummy": Start exporting action...
2023-02-10 18:02:11.142153 I Controller "Dummy 2x2": Setting new value of actuator
↳ "Actuator 1"...
2023-02-10 18:02:11.142153 S Controller "Dummy 2x2": Pseudo-export of an actuator.
↳ value: 1
2023-02-10 18:02:11.142153 I Controller "Dummy 2x2": Value of actuator "Actuator 1".
↳ set to 1
2023-02-10 18:02:11.142153 I Controller "Dummy 2x2": Setting new value of actuator
↳ "Actuator 2"...
2023-02-10 18:02:11.142153 S Controller "Dummy 2x2": Pseudo-export of an actuator.
↳ value: 2
2023-02-10 18:02:11.142153 I Controller "Dummy 2x2": Value of actuator "Actuator 2".
↳ set to 2
2023-02-10 18:02:11.142153 I System "Dummy": Waiting the system latency time of 1.0.
↳ seconds...
2023-02-10 18:02:12.158016 I System "Dummy": Start importing state...
2023-02-10 18:02:12.158016 I Controller "Dummy 2x2": Getting value of sensor "Sensor 1
↳ "...
2023-02-10 18:02:12.158016 S Controller "Dummy 2x2": Pseudo-import of a sensor value..
↳ .
2023-02-10 18:02:12.158016 I Controller "Dummy 2x2": Value of sensor "Sensor 1" = 0.
↳ 897198578560347
2023-02-10 18:02:12.158016 I Controller "Dummy 2x2": Getting value of sensor "Sensor 2
↳ "...
2023-02-10 18:02:12.158016 S Controller "Dummy 2x2": Pseudo-import of a sensor value..

```

(continues on next page)

(continued from previous page)

```

↪ .
2023-02-10 18:02:12.158016 I Controller "Dummy 2x2": Value of sensor "Sensor 2" = 0.
↪ 8543962410135211
2023-02-10 18:02:12.158016 I System "Dummy": Assessment for success...
2023-02-10 18:02:12.158016 I System "Dummy": Assessment for breakdown...
2023-02-10 18:02:12.158016 I System "Dummy": Assessment for success...
2023-02-10 18:02:12.158016 I System "Dummy": Assessment for breakdown...
2023-02-10 18:02:12.158016 I System "Dummy": Action processing finished successfully

```

**Cross Reference**

- [API Reference: Systems](#)

**Howto BF-SYSTEMS-011: Double Pendulum Systems wrapped with MuJoCo****Prerequisites**

Please install the following packages to run this examples properly:

- [MuJoCo](#)
- [lxml](#)
- [glfw](#)

**Executable code****Results**

The MuJoCo windows appears and shows the simulation of a pendulum system.

**Cross Reference**

- [API Reference: Systems](#)

**Howto BF-SYSTEMS-012: Cartpole Continuous Systems wrapped with MuJoCo****Prerequisites**

Please install the following packages to run this examples properly:

- [MuJoCo](#)
- [lxml](#)
- [glfw](#)

**Executable code****Results**

The MuJoCo windows appears and shows the simulation of a pendulum system.

**Cross Reference**

- [API Reference: Systems](#)

## Howto BF-SYSTEMS-013: MuJoCo Simulation with Camera

### Prerequisites

Please install the following packages to run this examples properly:

- [MuJoCo](#)
- [lxml](#)
- [glfw](#)

### Executable code

### Results

The MuJoCo window appears and shows the simulation of a random box on the table. A Matplotlib window appears also and shows the current image data from the camera.

### Cross Reference

- *API Reference: Systems*

## 10.1.5 Layer 4 - Machine Learning

### Basics

#### Howto BF-ML-001: Adaptive Model

### Executable code

```
## -----  
↪ -----  
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks  
## -- Package : mlpro.bf.examples  
## -- Module  : howto_bf_ml_001_adaptive_model.py  
## -----  
↪ -----  
## -- History :  
## -- yyyy-mm-dd  Ver.      Auth.      Description  
## -- 2023-02-15  1.0.0     DA         Creation  
## -----  
↪ -----  
  
  
  
Ver. 1.0.0 (2023-02-15)  
  
This module demonstrates the basic properties of an adaptive model in MLPro. It also ↪  
↪ gives an overview  
of all custom methods that can be used for own purposes.  
  
You will learn:  
  
1) How to implement a custom model  
  
2) How to let your model adapt on a dataset
```

(continues on next page)

(continued from previous page)

3) How to run your model as a task

```

"""

from mlpro.bf.ml import *

## -----
↪ -----
## -----
↪ -----
class MyModel (Model):

    # Please give your implementation a suitable name...
    C_NAME                = 'Demo'

    # If you want to use the visualization features you need to activate it explicitly..
↪ .
    C_PLOT_ACTIVE          = True

    # Any related scientific literature? Please add the bibliographic parameters. See ↪
↪ class
    # mlpro.bf.various.ScientificObject for further information...
    C_SCIREF_TYPE          = ScientificObject.C_SCIREF_TYPE_ARTICLE
    C_SCIREF_AUTHOR        = 'Dettef Arend, Mochammad Rizky Diprasetya, Steve Yuwono, ↪
↪ Andreas Schwung'
    C_SCIREF_TITLE         = 'MLPro - An integrative middleware framework for standardized ↪
↪ machine learning tasks in Python'
    C_SCIREF_JOURNAL       = 'Software Impacts'
    C_SCIREF_PUBLISHER     = 'Elsevier'
    C_SCIREF_YEAR          = '2022'
    C_SCIREF_VOLUME       = '14'
    C_SCIREF_DOI           = '10.1016/j.simpa.2022.100421'

## -----
↪ -----
    def _init_hyperparam(self, p_hp_layers : int, p_hp_neurons_per_layer : int ):
        """
        Define and initialize the specific hyperparameters of your model here. See ↪
↪ classes Model,
        HyperParamSpace, HyperParam, HyerParamTuple of sub-package mlpro.bf.ml for ↪
↪ further information.

        Parameters
        -----
        p_hp_layers : int
            Number of layers.
        p_hp_neurons_per_layer : int

```

(continues on next page)

(continued from previous page)

```

        Number of neurons per layer.
        """

        # 1 Define the hyperparameter space of your model...
        self._hyperparam_space.add_dim( HyperParam( p_name_short = 'number of layers',
                                                    p_base_set = Dimension.C_BASE_SET_N,
                                                    p_boundaries=[1,5]) )

        self._hyperparam_space.add_dim( HyperParam( p_name_short = 'neurons per layer',
                                                    p_base_set = Dimension.C_BASE_SET_N,
                                                    p_boundaries=[1,20]) )

        # 2 Initialize the hyperparameters on external values
        self._hyperparam_tuple = HyperParamTuple( p_set=self._hyperparam_space )
        self._hyperparam_tuple.set_values( [p_hp_layers, p_hp_neurons_per_layer] )

## -----
↳ -----
    def _adapt(self, p_dataset : list) -> bool:
        """
        This custom method is intended for explicit adaptation. Here you can specify
↳concrete
        adaptation data parameters and implement a suitable algorithm.

        Parameters
        -----
        p_dataset : list
            A dataset

        Returns
        -----
        adapted : bool
            True, if something has been adapted. False otherwise.
        """

        self.log(Log.C_LOG_TYPE_I, 'Incoming dataset:\n', p_dataset)
        self.log(Log.C_LOG_TYPE_I, 'My current hyperparameters are', self.get_
↳hyperparam().get_values() )
        self.log(Log.C_LOG_TYPE_W, 'By the way: I do not really adapt something;')
        return False

## -----
↳ -----
    def _adapt_on_event(self, p_event_id: str, p_event_object: Event) -> bool:
        """
        This custom method can be used to adapt something based on an event. The methods
↳needs to be
        registered as an event handler on a separate object that in turn raises the
↳event. See class
        mlpro.bf.events.EventHandler for further information. Context informations can

```

(continues on next page)



(continued from previous page)

```

↪ be extracted
    from the related event object in parameter p_event_object.

    Returns
    -----
    adapted : bool
        True, if something has been adapted. False otherwise.
    """

    self.log(Log.C_LOG_TYPE_I, 'Custom adaptation based on event', p_event_id)
    self.log(Log.C_LOG_TYPE_W, 'By the way: I do not really adapt something;')
    return False

## -----
↪ -----
    def _run(self, p_input : float):
        """
        This custom method implements the operational activities of your model. Here you
        ↪ can specify
        concrete runtime parameters to be processed.
        """

        self.log(Log.C_LOG_TYPE_I, 'Incoming data to be processed:', str(p_input))
        self.log(Log.C_LOG_TYPE_W, 'By the way: I do not really process something;')

## -----
↪ -----
    def get_accuracy(self) -> float:
        """
        This custom method returns the current accuracy of your model...

        Returns
        -----
        accuracy : float
            Accuracy of the model as a scalar value in interval [0,1]
        """

        return 0.0

## -----
↪ -----
    def _init_plot_2d(self, p_figure: Figure, p_settings: PlotSettings):
        """
        This custom method is intended to prepare a 2-dimensional plot. See classes
        ↪ PlotSettings and
        Plottable of sub-package mlpro.bf.plot or the online documentation for further
        ↪ information.
        """

```

(continues on next page)

(continued from previous page)

```

self.log(Log.C_LOG_TYPE_W, 'My specific plot initialization...')
return super()._init_plot_2d(p_figure, p_settings)

## -----
↪ -----
def _init_plot_3d(self, p_figure: Figure, p_settings: PlotSettings):
    """
    This custom method is intended to prepare a 3-dimensional plot. See classes_
    ↪ PlotSettings and
    Plottable of sub-package mlpro.bf.plot or the online documentation for further_
    ↪ information.
    """

    self.log(Log.C_LOG_TYPE_W, 'My specific plot initialization...')
    return super()._init_plot_3d(p_figure, p_settings)

## -----
↪ -----
def _init_plot_nd(self, p_figure: Figure, p_settings: PlotSettings):
    """
    This custom method is intended to prepare a n-dimensional plot. See classes_
    ↪ PlotSettings and
    Plottable of sub-package mlpro.bf.plot or the online documentation for further_
    ↪ information.
    """

    self.log(Log.C_LOG_TYPE_W, 'My specific plot initialization...')
    return super()._init_plot_nd(p_figure, p_settings)

## -----
↪ -----
def _update_plot_2d(self, p_settings: PlotSettings, **p_kwargs):
    """
    This custom method is intended to update your 2-dimensional plot. See classes_
    ↪ PlotSettings and
    Plottable of sub-package mlpro.bf.plot or the online documentation for further_
    ↪ information.
    """

    self.log(Log.C_LOG_TYPE_W, 'My specific plot updates...')
    return super()._update_plot_2d(p_settings, **p_kwargs)

## -----
↪ -----
def _update_plot_3d(self, p_settings: PlotSettings, **p_kwargs):
    """
    This custom method is intended to update your 3-dimensional plot. See classes_
    ↪ PlotSettings and

```

(continues on next page)

(continued from previous page)

```

        Plottable of sub-package mlpro.bf.plot or the online documentation for further
        ↪information.
        """

        self.log(Log.C_LOG_TYPE_W, 'My specific plot updates...')
        return super()._update_plot_3d(p_settings, **p_kwargs)

## -----
↪-----
    def _update_plot_nd(self, p_settings: PlotSettings, **p_kwargs):
        """
        This custom method is intended to update your n-dimensional plot. See classes
        ↪PlotSettings and
        Plottable of sub-package mlpro.bf.plot or the online documentation for further
        ↪information.
        """

        self.log(Log.C_LOG_TYPE_W, 'My specific plot updates...')
        return super()._update_plot_nd(p_settings, **p_kwargs)

# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    visualize = True
    logging = Log.C_LOG_ALL

else:
    # 1.2 Parameters for internal unit test
    visualize = False
    logging = Log.C_LOG_NOTHING

# 2 Instantiation of your custom model (multitasking is disabled)
mymodel = MyModel( p_range_max = Range.C_RANGE_NONE,
                   p_visualize = visualize,
                   p_logging = logging,
                   p_hp_layers = 5,
                   p_hp_neurons_per_layer = 10 )

# 3 Pseudo-adaptation based on a dataset
ds = [ (1,2), (3,4), (5,6), (6,7) ]
mymodel.adapt( p_dataset = ds )

# 4 Now we let your model do it's job
mymodel.run( p_input = 42 )

```

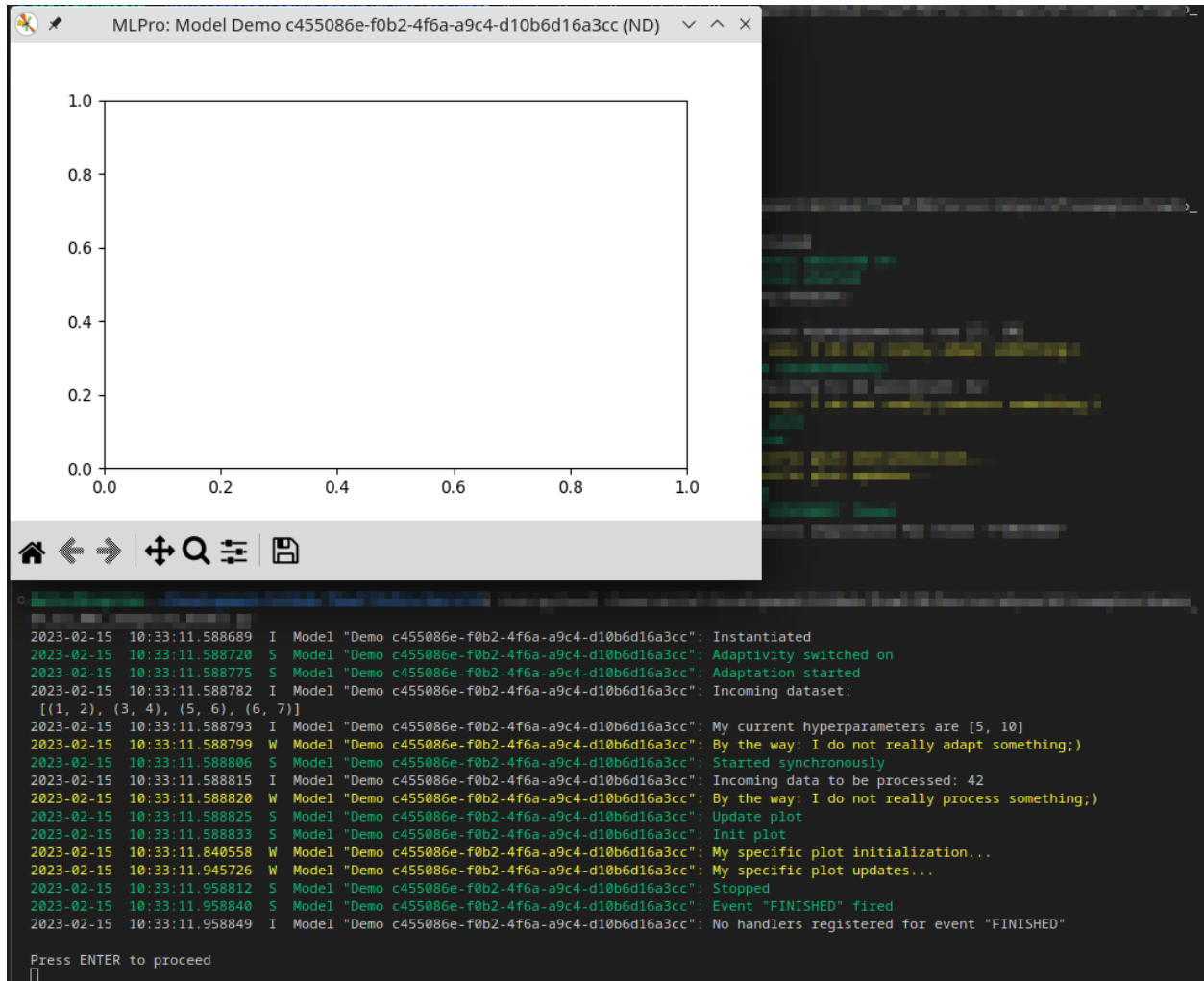
(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    input( '\nPress ENTER to proceed\n')
```

## Results

As shown below, the howto logs all steps and a demo window for visualization appears...



## Cross Reference

- [API Reference: Machine Learning](#)

## Howto BF-ML-010: Hyperparameters

### Executable code

```
## -----
↪-----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_ml_010_hyperparameters.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2021-08-31  0.0.0     SY         Creation
## -- 2021-09-01  1.0.0     SY         Release of first version
## -- 2021-09-11  1.0.0     MRD        Change Header information to match our new library.↪
↪name
## -- 2021-12-10  1.0.1     DA         Refactoring, little beautifying
## -- 2022-02-25  1.0.2     SY         Refactoring due to auto generated ID in class.↪
↪Dimension
## -- 2022-10-12  1.0.3     DA         Renaming/refactoring
## -- 2023-02-15  1.1.0     DA         Renaming
## -- 2023-03-02  1.1.1     LSB        Refactoring
## -----
↪-----
"""
Ver. 1.1.1 (2023-03-02)

This module demonstrates how to set-up hyperparameters using available HyperParamTuple,
HyperParamSpace, and HyperParam classes.

You will learn:

1. How to use the Hyperparameter class of MLPro and its functionalities in Native and↪
↪custom implementations.

2. How to create hyperparameter space and add dimensions to the space.

3. How to create and set values for a hyperparameter tuple.
"""

from mlpro.bf.ml import *

# 1 Setup a class that requires a tuple of hyperparameters
class MyHyperparameter:

    def __init__(self):
        # 1.1 Construct a hyperparameter space using HyperParamSpace() and an empty↪
↪tuple
```

(continues on next page)

(continued from previous page)

```

self._hyperparam_space = HyperParamSpace()
self._hyperparam_tuple = None
self._init_hyperparam()

def _init_hyperparam(self):
    # 1.2 Declare hyperparameters with unique id, names, and data type
    self._hyperparam_space.add_dim(HyperParam('num_states', 'Z'))
    self._hyperparam_space.add_dim(HyperParam('smoothing', 'R'))
    self._hyperparam_space.add_dim(HyperParam('lr_rate', 'R'))
    self._hyperparam_space.add_dim(HyperParam('buffer_size', 'Z'))
    self._hyperparam_space.add_dim(HyperParam('update_rate', 'Z'))
    self._hyperparam_space.add_dim(HyperParam('sampling_size', 'Z'))
    self._hyperparam_tuple = HyperParamTuple(self._hyperparam_space)

    # 1.3 Set the hyperparameter with a default value
    ids_ = self._hyperparam_tuple.get_dim_ids()
    self._hyperparam_tuple.set_value(ids_[0], 100)
    self._hyperparam_tuple.set_value(ids_[1], 0.035)
    self._hyperparam_tuple.set_value(ids_[2], 0.0001)
    self._hyperparam_tuple.set_value(ids_[3], 100000)
    self._hyperparam_tuple.set_value(ids_[4], 100)
    self._hyperparam_tuple.set_value(ids_[5], 256)

def get_hyperparam(self) -> HyperParamTuple:
    return self._hyperparam_tuple

if __name__ == "__main__":
    printing = True
else:
    printing = False

# 2 Get value from the hyperparameter tuple
myParameter = MyHyperparameter()
for idx in myParameter.get_hyperparam().get_dim_ids():
    par_name = myParameter.get_hyperparam().get_related_set().get_dim(idx).get_name_
    ↪short()
    par_val = myParameter.get_hyperparam().get_value(idx)
    if printing: print('Variable with ID %s = %.2f'%(par_name, par_val))

# 3 Overwrite current value with new desired value
ids_ = myParameter.get_hyperparam().get_dim_ids()
myParameter.get_hyperparam().set_value(ids_[0], 50)

if printing:
    print('\nA new value for variable ID ids_[0]')

```

(continues on next page)

(continued from previous page)

```
print('Variable with ID ids_[0] = %.2f'%(myParameter.get_hyperparam().get_value(ids_
↪[0])))
```

## Results

```
Variable with ID num_states = 100.00
Variable with ID smoothing = 0.04
Variable with ID lr_rate = 0.00
Variable with ID buffer_size = 100000.00
Variable with ID update_rate = 100.00
Variable with ID sampling_size = 256.00
```

```
A new value for variable ID ids_[0]
Variable with ID ids_[0] = 50.00
```

## Cross Reference

- *API Reference: Machine Learning*

## Adaptive State-based Systems

## 10.2 MLPro-SL - Supervised Learning

Coming soon...

## 10.3 MLPro-RL - Reinforcement Learning

The following examples demonstrate various functionalities of MLPro-RL:

### 10.3.1 Elementary or Uncategorized Topics

#### Howto RL-001: Reward

#### Executable code

```
## -----
↪ -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_001_reward.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2021-05-30  0.0.0     DA         Creation
## -- 2021-05-31  1.0.0     DA         Release of first version
## -- 2021-09-11  1.0.1     MRD        Change Header information to match our new library.
↪ name
## -- 2022-10-13  1.0.2     SY         Refactoring
```

(continues on next page)

(continued from previous page)

```

## -- 2023-03-02  1.0.3      LSB      Refactoring
## -----
↪-----

"""
Ver. 1.0.3 (2023-03-02)

This module shows how to create and interpret reward objects in own projects.

You will learn:

1. How to use the reward class of MLPro.

2. How to use reward class for different reward types supported in MLPro.

"""

from mlpro.bf.various import Log
from mlpro.rl import Reward


class MyLog(Log):
    C_TYPE      = 'Reward Demo'
    C_NAME      = ''

if __name__ == "__main__":
    # 1 Some initial stuff
    my_log = MyLog()

    # 1.1 Unique agent ids
    C_AGENT_1   = 1
    C_AGENT_2   = 2
    C_AGENT_3   = 3

    # 1.2 Unique action ids
    C_AGENT_1_ACT_1 = 1
    C_AGENT_1_ACT_2 = 2
    C_AGENT_1_ACT_3 = 3
    C_AGENT_2_ACT_1 = 4

    # 2 Rewards as single overall scalar values (independent from agents and actions)
    my_log.log(Log.C_LOG_TYPE_I, 'Example for reward type C_TYPE_OVERALL:')
    reward = Reward(p_type=Reward.C_TYPE_OVERALL)
    reward.set_overall_reward(4.77)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward is just a scalar...', reward.get_agent_
↪reward(0), '\n')

    # 3 Rewards as scalar values for every agent

```

(continues on next page)



(continued from previous page)

```

my_log.log(Log.C_LOG_TYPE_I, 'Example for reward type C_TYPE_EVERY_AGENT')
reward = Reward(p_type=Reward.C_TYPE_EVERY_AGENT)
my_log.log(Log.C_LOG_TYPE_I, 'Reward is a list with entries for each agent...')
reward.add_agent_reward(C_AGENT_1, 4.77)
my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 1 added:', reward.get_agent_reward(C_
↪AGENT_1))
reward.add_agent_reward(C_AGENT_2, 5.19)
my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 2 added:', reward.get_agent_reward(C_
↪AGENT_2))
reward.add_agent_reward(C_AGENT_3, 0.23)
my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 3 added:', reward.get_agent_reward(C_
↪AGENT_3), '\n')

# 4 Rewards as scalar values for every agent and it's actions
my_log.log(Log.C_LOG_TYPE_I, 'Example for reward type C_TYPE_EVERY_ACTION')
reward = Reward(p_type=Reward.C_TYPE_EVERY_ACTION)
my_log.log(Log.C_LOG_TYPE_I, 'Reward is a list with entries for each agent and its_
↪action components...')
reward.add_action_reward(C_AGENT_1, C_AGENT_1_ACT_1, 1.23)
my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 1, action 1 added:', reward.get_
↪action_reward(C_AGENT_1, C_AGENT_1_ACT_1))
reward.add_action_reward(C_AGENT_1, C_AGENT_1_ACT_2, 0.47)
my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 1, action 2 added:', reward.get_
↪action_reward(C_AGENT_1, C_AGENT_1_ACT_2))
reward.add_action_reward(C_AGENT_1, C_AGENT_1_ACT_3, 1.63)
my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 1, action 3 added:', reward.get_
↪action_reward(C_AGENT_1, C_AGENT_1_ACT_3))
reward.add_action_reward(C_AGENT_2, C_AGENT_2_ACT_1, 4.23)
my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 2, action 4 added:', reward.get_
↪action_reward(C_AGENT_2, C_AGENT_2_ACT_1))

```

**Cross Reference**

- [API Reference - RL Agent](#)
- [API Reference - RL Environments](#)

**10.3.2 Environments****Howto RL-ENV-003: Run Agent with random Policy in Double Pendulum MuJoCo Environment****Prerequisites**

Please install the following packages to run this examples properly:

- NumPy
- MuJoCo

**Executable code**

```

## -----
↪-----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks

```

(continues on next page)

(continued from previous page)

```

## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_env_003_run_agent_with_random_policy_on_double_pendulum_mujoco_
↳environment.py
## -----
↳-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2022-09-17  0.0.0     MRD        Creation
## -- 2022-12-11  0.0.1     MRD        Refactor due to new bf.Systems
## -- 2022-12-11  1.0.0     MRD        First Release
## -- 2023-01-07  1.0.1     MRD        Add State Mapping between MuJoCo model and↳
↳Environment State Space
## -- 2023-01-27  1.1.0     MRD        Implement Pendulum Environment, refactor due to↳
↳different MuJoCo
## --                                     mechanism
## -- 2023-02-13  1.1.1     MRD        Refactor
## -- 2023-02-23  1.2.0     DA         Renamed
## -- 2024-02-16  1.2.1     SY         Renaming Module
## -----
↳-----

"""
Ver. 1.2.1 (2024-02-16)

This module shows how to run a random policy on Double Pendulum with MuJoCo Simulation.

You will learn:

1) How to set up an own agent using MLPro's builtin random actions policy
2) How to set up an own RL scenario including your agent and MLPro's double pendulum↳
↳environment
3) How to integrate MuJoCo as the Simulation
4) How to reset and run your own scenario

"""

import random
import numpy as np
import os

import mlpro
from mlpro.bf.ml import Model
from mlpro.bf.ops import Mode
from mlpro.bf.various import Log
from mlpro.rl.models_agents import Policy, Agent
from mlpro.rl.models_train import RLScenario
from mlpro.bf.systems import State, Action
from mlpro.rl.models_env_ada import SARSElement

```

(continues on next page)

(continued from previous page)

```

from mlpro.rl.models_env import Environment
from mlpro.rl.models_agents import Reward
from mlpro.bf.systems import *

# 1 Implement the Environment
class PendulumEnvironment (Environment):

    C_NAME          = 'PendulumEnvironment'
    C_REWARD_TYPE   = Reward.C_TYPE_OVERALL

    def __init__(self,
                  p_mode=Mode.C_MODE_SIM,
                  p_mujoco_file=None,
                  p_frame_skip: int = 1,
                  p_state_mapping=None,
                  p_action_mapping=None,
                  p_camera_conf: tuple = (None, None, None),
                  p_visualize: bool = False,
                  p_logging=Log.C_LOG_ALL):

        super().__init__(p_mode=p_mode,
                         p_mujoco_file=p_mujoco_file,
                         p_frame_skip=p_frame_skip,
                         p_state_mapping=p_state_mapping,
                         p_action_mapping=p_action_mapping,
                         p_camera_conf=p_camera_conf,
                         p_visualize=p_visualize,
                         p_logging=p_logging)

        self._state = State(self._state_space)
        self.reset()

    def _compute_reward(self, p_state_old: State = None, p_state_new: State = None) ->
↳ Reward:
        reward = Reward(self.C_REWARD_TYPE)
        reward.set_overall_reward(1)
        return reward

    def _reset(self, p_seed=None) -> None:
        pass

# 1 Implement your own agent policy
class MyPolicy (Policy):

    C_NAME          = 'MyPolicy'

    def set_random_seed(self, p_seed=None):
        random.seed(p_seed)

```

(continues on next page)

(continued from previous page)

```

def compute_action(self, p_state: State) -> Action:
    # 1.1 Create a numpy array for your action values
    my_action_values = np.zeros(self._action_space.get_num_dim())

    # 1.2 Computing action values is up to you...
    for d in range(self._action_space.get_num_dim()):
        my_action_values[d] = np.random.uniform(-50, 50)

    # 1.3 Return an action object with your values
    return Action(self._id, self._action_space, my_action_values)

def _adapt(self, p_sars_elem: SARSElement) -> bool:
    # 1.4 Adapting the internal policy is up to you...
    self.log(self.C_LOG_TYPE_W, 'Sorry, I am a stupid agent...')

    # 1.5 Only return True if something has been adapted...
    return False

# 2 Implement your own RL scenario
class MyScenario (RLScenario):

    C_NAME      = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize:bool, p_logging) -> Model:
        # 2.1 Setup environment
        model_file = os.path.join(os.path.dirname(mlpro.__file__), "bf/systems/pool/
↪mujoco", "doublependulum.xml")
        self._env = PendulumEnvironment(p_logging=logging, p_mujoco_file=model_file, p_
↪visualize=visualize)

        # 2.2 Setup standard single-agent with own policy
        return Agent( p_policy=MyPolicy( p_observation_space=self._env.get_state_space(),
                                         p_action_space=self._env.get_action_space(),
                                         p_buffer_size=1,
                                         p_ada=p_ada,
                                         p_visualize=p_visualize,
                                         p_logging=p_logging),

                    p_envmodel=None,
                    p_name='Smith',
                    p_ada=p_ada,
                    p_visualize=p_visualize,
                    p_logging=p_logging)

# 3 Create scenario and run some cycles
if __name__ == "__main__":
    # 3.1 Parameters for demo mode
    cycle_limit = 2000
    logging      = Log.C_LOG_ALL

```

(continues on next page)

(continued from previous page)

```

    visualize    = True

else:
    # 3.2 Parameters for internal unit test
    cycle_limit = 10
    logging      = Log.C_LOG_NOTHING
    visualize    = False

# 3.3 Create your scenario and run some cycles
myscenario = MyScenario(
    p_mode=Mode.C_MODE_SIM,
    p_ada=True,
    p_cycle_limit=cycle_limit,
    p_visualize=visualize,
    p_logging=logging
)

myscenario.reset(p_seed=3)

myscenario.run()

```

**Results**

Running this howto opens a new MuJoCo window that shows the double pendulum environment controlled by a random policy. This, in turn, causes chaotic behavior.

**Cross Reference**

- [API Reference - RL Agent](#)
- [API Reference - RL Environments](#)
- [API Reference - RL Scenario and Training](#)

**10.3.3 Adaptive Environments**

**Howto RL-AE-001: Available Soon**

**10.3.4 Model-based Reinforcement Learning**

**Howto RL-MB-001: Train a model-based Agent on native Grid World Environment**

**Prerequisites**

Please install the following packages to run this examples properly:

- [PyTorch](#)

**Executable code**

```

## -----
↪ -----

```

(continues on next page)

(continued from previous page)

```

## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_mb_001_grid_world_environment.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2022-09-19  0.0.0     SY         Creation
## -- 2022-10-06  1.0.0     SY         Release first version
## -- 2022-10-07  1.0.1     SY         Add plotting
## -- 2022-10-13  1.0.2     SY         Refactoring
## -- 2022-11-07  1.1.0     DA         Refactoring
## -- 2023-02-02  1.2.0     DA         Refactoring
## -- 2023-02-04  1.2.1     SY         Add multiprocessing functionality and refactoring
## -- 2023-02-10  1.2.2     SY         Switch multiprocessing to threading
## -- 2023-03-07  2.0.0     SY         Update due to MLPro-SL
## -- 2023-03-08  2.0.1     SY         Refactoring
## -- 2023-03-10  2.0.2     SY         Refactoring
## -- 2023-05-04  2.0.3     SY         Refactoring
## -- 2023-05-04  2.0.4     LSB        Refactoring
## -- 2024-02-16  2.0.5     SY         Renaming Module
## -----
↪ -----

"""
Ver. 2.0.5 (2024-02-16)

This module shows how to incorporate MPC in Model-Based RL on Grid World problem as well_
↪ as using
PyTorch-based MLP network from MLPro-SL's pool of objects.

You will learn:

1) How to set up an own agent on Grid World problem
2) How to set up model-based RL (MBRL) training using network from MLPro-SL's pool
3) How to incorporate MPC into MBRL training
4) How to use multiprocessing on MPC

"""

import torch
import numpy as np
from mlpro.bf.plot import DataPlotting
from mlpro.bf.math import *
from mlpro.rl import *
from mlpro.rl.models import *
from mlpro.rl.pool.envs.gridworld import *
from mlpro.rl.pool.policies.randomgenerator import RandomGenerator

```

(continues on next page)

(continued from previous page)

```

from mlpro.sl.pool.afct.fnn.pytorch.mlp import PyTorchMLP
from pathlib import Path
from mlpro.rl.pool.actionplanner.mpc import MPC
from mlpro.rl.pool.envs.gridworld import *
import mlpro.bf.mt as mt

## -----
↪ -----
## -----
↪ -----

# 1 Implement the random RL scenario
class ScenarioGridWorld(RLScenario):

    C_NAME      = 'Grid World with Random Actions'

## -----
↪ -----

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        self._env = GridWorld(p_logging=p_logging,
                               p_action_type=GridWorld.C_ACTION_TYPE_DISC_2D,
                               p_max_step=100)

        # 1.2 Setup and return random action agent
        policy_random = RandomGenerator(p_observation_space=self._env.get_state_space(),
                                         p_action_space=self._env.get_action_space(),
                                         p_buffer_size=1,
                                         p_ada=1,
                                         p_logging=p_logging)

        # Setup Adaptive Function
        afct_strans = AFctSTrans(
            PyTorchMLP,
            p_state_space=self._env._state_space,
            p_action_space=self._env._action_space,
            p_threshold=0.1,
            p_buffer_size=100,
            p_batch_size=10,
            p_ada=p_ada,
            p_logging=p_logging,
            p_update_rate=1,
            p_num_hidden_layers=3,
            p_hidden_size=128,
            p_activation_fct=torch.nn.ReLU(),
            p_output_activation_fct=torch.nn.ReLU(),

```

(continues on next page)

(continued from previous page)

```

        p_optimizer=torch.optim.Adam,
        p_loss_fct=torch.nn.MSELoss,
        p_learning_rate=3e-4
    )

    envmodel = EnvModel(
        p_observation_space=self._env._state_space,
        p_action_space=self._env._action_space,
        p_latency=self._env.get_latency(),
        p_afct_strans=afct_strans,
        p_afct_reward=self._env,
        p_afct_success=self._env,
        p_afct_broken=self._env,
        p_ada=p_ada,
        p_init_states=self._env.get_state(),
        p_logging=p_logging
    )

    mb_training_param = dict(p_cycle_limit=100,
                             p_cycles_per_epi_limit=100,
                             p_max_stagnations=0,
                             p_collect_states=False,
                             p_collect_actions=False,
                             p_collect_rewards=False,
                             p_collect_training=False)

    return Agent(
        p_policy=policy_random,
        p_envmodel=envmodel,
        p_em_acc_thsld=0.8,
        p_action_planner=MPC(p_range_max=mt.Async.C_RANGE_THREAD,
                             p_logging=p_logging),
        p_predicting_horizon=5,
        p_controlling_horizon=1,
        p_planning_width=50,
        p_name='Smith',
        p_ada=p_ada,
        p_visualize=p_visualize,
        p_logging=p_logging,
        **mb_training_param
    )

## -----
↪ -----
## -----
↪ -----

# 2 Train agent in scenario

```

(continues on next page)



(continued from previous page)

```

if __name__ == "__main__":
    # 2.1 Parameters for demo mode
    cycle_limit = 5000
    logging      = Log.C_LOG_ALL
    visualize    = True
    path         = str(Path.home())
    plotting     = True
else:
    # 2.2 Parameters for internal unit test
    cycle_limit = 10
    logging      = Log.C_LOG_NOTHING
    visualize    = False
    path         = None
    plotting     = False

training = RLTraining(
    p_scenario_cls=ScenarioGridWorld,
    p_cycle_limit=cycle_limit,
    p_cycles_per_epi_limit=100,
    p_collect_states=True,
    p_collect_actions=True,
    p_collect_rewards=True,
    p_collect_training=True,
    p_path=path,
    p_logging=logging,
)

training.run()

## -----
↪-----
## -----
↪-----

# 3 Plotting with MLpro
class MyDataPlotting(DataPlotting):

## -----
↪-----

    def get_plots(self):
        """
        A function to plot data
        """
        for name in self.data.names:
            maxval = 0
            minval = 0
            if self.printing[name][0]:

```

(continues on next page)

(continued from previous page)

```

fig = plt.figure(figsize=(7, 7))
raw = []
label = []
ax = fig.subplots(1, 1)
ax.set_title(name)
ax.grid(True, which="both", axis="both")
for fr_id in self.data.frame_id[name]:
    raw.extend(self.data.get_values(name, fr_id))
    if self.printing[name][1] == -1:
        maxval = max(raw)
        minval = min(raw)
    else:
        maxval = self.printing[name][2]
        minval = self.printing[name][1]

    label.append("%s" % fr_id)
ax.plot(raw)
ax.set_ylim(minval - (abs(minval) * 0.1), maxval + (abs(maxval) * 0.1))
plt.xlabel("continuous cycles")
self.plots[0].append(name)
self.plots[1].append(ax)
if self.showing:
    plt.show()
else:
    plt.close(fig)

## -----
↪-----
## -----
↪-----

if __name__ == "__main__":
    data_printing = {
        "Cycle": [False],
        "Day": [False],
        "Second": [False],
        "Microsecond": [False],
        "Smith": [True, -1],
    }

    mem = training.get_results().ds_rewards
    mem_plot = MyDataPlotting(mem, p_showing=plotting, p_printing=data_printing)
    mem_plot.get_plots()

```

## Results

After the environment is initiated, the training will run for the specified amount of limits. The expected initial console output can be seen below.

```

YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -----
↪-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -- Training episode 0 started...
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -----
↪-----

YYYY-MM-DD HH:MM:SS.SSSSSS I Agent "": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S Agent "": Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -----
↪-----

YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -----
↪-----

YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -- Training run 0 started...
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -----
↪-----

YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -----
↪-----

YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -----
↪-----

YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -- Training episode 0 started...
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "RL": -----
↪-----

YYYY-MM-DD HH:MM:SS.SSSSSS I Agent "": Action computation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Agent "": Action computation finished
YYYY-MM-DD HH:MM:SS.SSSSSS S Agent "": Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Agent "": Action computation started
...

```

After termination the local result folder contains the training result files:

- agent\_actions.csv
- env\_rewards.csv
- env\_states.csv
- evaluation.csv
- summary.csv
- trained model.pkl

### Cross Reference

- [API Reference - RL Agent](#)
- [API Reference - RL Environments](#)
- [API Reference - Environment Model](#)
- [API Reference - RL Scenario and Training](#)

### 10.3.5 3rd Party Support

Discover numerous additional examples in combination with popular *3rd party packages*.

## 10.4 MLPro-GT - Game Theory

The following examples demonstrate various functionalities of MLPro-GT:

### 10.4.1 Dynamic Games

#### Howto GT-DG-001: Run Multi-Player with Own Policy

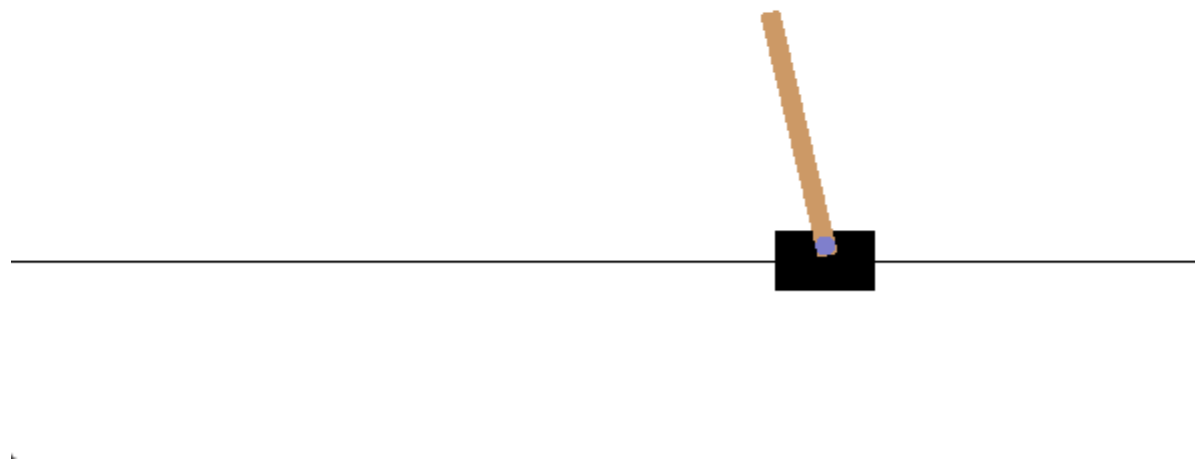
##### Prerequisites

Please install the following packages to run this examples properly:

- NumPy
- OpenAI Gym

##### Executable code

##### Results



Three Gym Cartpole game board windows should appear and the following output should be expected in the console.

```

YYYY-MM-DD HH:MM:SS.SSSSSS I Potential Game Board MultiCartPole(PGT): Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I Potential Game Board MultiCartPole(PGT): Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I Potential Game Board MultiCartPole(PGT): Operation mode_
↪set to 0
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1": Operation mode set to 0
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1": Operation mode set to 0
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1": Operation mode set to 0
YYYY-MM-DD HH:MM:SS.SSSSSS I Potential Game Board MultiCartPole(PGT): Reset
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1" (0): Reset
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1" (1): Reset
YYYY-MM-DD HH:MM:SS.SSSSSS I OpenAI Gym Env Env "CartPole-v1" (2): Reset
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy MyPolicy: Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Player Neo: Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I Player Neo: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Player Neo: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: Player 0 Neo added.
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy MyPolicy: Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Player Trinity: Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I Player Trinity: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Player Trinity: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: Player 1 Trinity added.
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Matrix: Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Matrix: Operation mode set to 0
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Matrix: Process time 0:00:00 Start of processing
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Matrix: Process time 0:00:00 : Start of cycle 0
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Matrix: Process time 0:00:00 : Agent computes_
↪action...
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: Start of action computation_
↪for all agents...
YYYY-MM-DD HH:MM:SS.SSSSSS I Player 0 Neo: Action computation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Player 0 Neo: Action computation finished
YYYY-MM-DD HH:MM:SS.SSSSSS I Player 1 Trinity: Action computation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Player 1 Trinity: Action computation finished
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: End of action computation for_
↪all agents...
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Matrix: Process time 0:00:00 : Env processes action.
↪...
YYYY-MM-DD HH:MM:SS.SSSSSS I Potential Game Board MultiCartPole(PGT): Start_
↪processing action

```

(continues on next page)

(continued from previous page)

```

YYYY-MM-DD HH:MM:SS.SSSSSS I Potential Game Board MultiCartPole(PGT): Actions of
↪agent 0 = [0.02821633]
YYYY-MM-DD HH:MM:SS.SSSSSS I Potential Game Board MultiCartPole(PGT): Actions of
↪agent 1 = [0.5796828 0.73351315]
YYYY-MM-DD HH:MM:SS.SSSSSS I Potential Game Board MultiCartPole(PGT): Action
↪processing finished successfully
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Matrix: Process time 0:00:01 : Agent adapts policy..
↪.
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: Start of adaptation for all
↪agents...
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: Start adaption for agent 0
YYYY-MM-DD HH:MM:SS.SSSSSS I Player 0 Neo: Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy MyPolicy: Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS W Policy MyPolicy: Sorry, I am a stupid agent...
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: Start adaption for agent 1
YYYY-MM-DD HH:MM:SS.SSSSSS I Player 1 Trinity: Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy MyPolicy: Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS W Policy MyPolicy: Sorry, I am a stupid agent...
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: End of adaptation for all
↪agents...
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Matrix: Process time 0:00:01 : End of cycle 0
...
YYYY-MM-DD HH:MM:SS.SSSSSS I Multi-Player Human Beings: Start vizualization for all
↪agents...
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Matrix: Process time 0:00:12 End of processing

```

## Cross Reference

- *API Reference: Game Theory in Dynamic Games*

## Howto GT-DG-002: Train Multi-Player

### Prerequisites

Please install the following packages to run this examples properly:

- NumPy
- OpenAI Gym

### Executable code

**Results** After the multiple game boards are initialised, the console will be filled with training logs and the final training result should show up at the end of the script.

```

YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -----
↪-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Training Results of run 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -----
↪-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -----
↪-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Scenario           : Game Matrix

```

(continues on next page)

(continued from previous page)

```

YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Model           : Multi-Player Human
↪Beings
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Start time stamp : YYYY-MM-DD HH:MM:SS.
↪SSSSSS
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- End time stamp   : YYYY-MM-DD HH:MM:SS.
↪SSSSSS
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Duration        : 0:00:12.329561
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Start cycle id   : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- End cycle id     : 199
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Training cycles  : 200
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Evaluation cycles : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Adaptations      : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- High score        : None
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Results stored in : "C:\Users\%username%\
↪YYYY-MM-DD HH:MM:SS Training GT"
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Training Episodes : 14
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -- Evaluations        : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -----
↪-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results RL: -----
↪-----

```

**Cross Reference**

- *API Reference: Game Theory in Dynamic Games*

**Howto GT-DG-003: Train Multi-Player in Potential Games****Prerequisites**

Please install the following packages to run this examples properly:

- NumPy

**Executable code****Results**

```

YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↪-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↪-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -- Training run 0 started...
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↪-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↪-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↪-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -- Training episode 0 started...
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↪-----

```

(continues on next page)

(continued from previous page)

```

YYYY-MM-DD HH:MM:SS.SSSSSS E GT Game "Matrix": Process time 0:01:40 : Environment_
↳ terminated
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": Limit of 100 cycles per episode_
↳ reached (Environment)
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -- Training episode 0 finished_
↳ after 100 cycles
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -- Training cycles finished: 100
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳ -----

YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -- Training episode 1 started...
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳ -----

YYYY-MM-DD HH:MM:SS.SSSSSS E GT Game "Matrix": Process time 0:01:40 : Environment_
↳ terminated
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": Limit of 100 cycles per episode_
↳ reached (Environment)
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -- Training episode 1 finished_
↳ after 100 cycles
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -- Training cycles finished: 200
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳ -----

YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": Training cycle limit 200 reached
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -- Training run 0 finished
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳ -----

YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Results stored in : "C:\Users\%username
↳ %\YYYY-MM-DD HH:MM:SS Training GT"
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Training Results of run 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -----
↳ -----

```

(continues on next page)



(continued from previous page)

```

YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Scenario      : GT Game Matrix
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Model        : GT Multi-Player
↳ BGLP Players with Random Policies
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Start time stamp : YYYY-MM-DD
↳ HH:MM:SS.SSSSSS
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- End time stamp  : YYYY-MM-DD
↳ HH:MM:SS.SSSSSS
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Duration       : 0:00:01.664187
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Start cycle id  : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- End cycle id    : 199
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Training cycles  : 200
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Evaluation cycles : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Adaptations      : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- High score       : None
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Training Episodes : 2
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Evaluations      : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -----
↳ -----

```

### Cross Reference

- *API Reference: Game Theory in Dynamic Games*

## Howto GT-DG-004: Train Multi-Player in Stackelberg Games

### Prerequisites

Please install the following packages to run this examples properly:

- NumPy

### Executable code

### Results

```

YYYY-MM-DD HH:MM:SS.SSSSSS I Training "GT Training": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Game "Matrix": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Reset
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Multi-Player SG "BGLP Players with Random Policies":
↳ Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy "MyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 478c892f-f980-4f5a-b894-f6d57dd357f7":
↳ Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "BELT_CONVEYOR_A (Leader)": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "BELT_CONVEYOR_A (Leader) 478c892f-f980-
↳ 4f5a-b894-f6d57dd357f7": Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 478c892f-f980-4f5a-b894-f6d57dd357f7":
↳ Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "BELT_CONVEYOR_A (Leader) 478c892f-f980-
↳ 4f5a-b894-f6d57dd357f7": Adaptivity switched on

```

(continues on next page)

(continued from previous page)

```

YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 478c892f-f980-4f5a-b894-f6d57dd357f7":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
↳GT Player SG BELT_CONVEYOR_A (Leader) 478c892f-f980-4f5a-b894-f6d57dd357f7 added.
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy "MyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 61ceca2b-52fc-4f34-8fbb-36bff56d9e1e":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "VACUUM_PUMP_B (Follower)": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "VACUUM_PUMP_B (Follower) 61ceca2b-52fc-
↳4f34-8fbb-36bff56d9e1e": Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 61ceca2b-52fc-4f34-8fbb-36bff56d9e1e":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "VACUUM_PUMP_B (Follower) 61ceca2b-52fc-
↳4f34-8fbb-36bff56d9e1e": Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 61ceca2b-52fc-4f34-8fbb-36bff56d9e1e":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
↳GT Player SG VACUUM_PUMP_B (Follower) 61ceca2b-52fc-4f34-8fbb-36bff56d9e1e added.
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy "MyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 6b4f1ea8-9b24-48b1-b4bc-f58662dc9cef":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "VIBRATORY_CONVEYOR_B (Follower)":
↳Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "VIBRATORY_CONVEYOR_B (Follower) 6b4f1ea8-
↳9b24-48b1-b4bc-f58662dc9cef": Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 6b4f1ea8-9b24-48b1-b4bc-f58662dc9cef":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "VIBRATORY_CONVEYOR_B (Follower) 6b4f1ea8-
↳9b24-48b1-b4bc-f58662dc9cef": Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 6b4f1ea8-9b24-48b1-b4bc-f58662dc9cef":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
↳GT Player SG VIBRATORY_CONVEYOR_B (Follower) 6b4f1ea8-9b24-48b1-b4bc-f58662dc9cef
↳added.
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy "MyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 2aef81a6-135c-4ce5-9b47-01576e635930":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "VACUUM_PUMP_C (Follower)": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "VACUUM_PUMP_C (Follower) 2aef81a6-135c-
↳4ce5-9b47-01576e635930": Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 2aef81a6-135c-4ce5-9b47-01576e635930":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "VACUUM_PUMP_C (Follower) 2aef81a6-135c-
↳4ce5-9b47-01576e635930": Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 2aef81a6-135c-4ce5-9b47-01576e635930":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
↳GT Player SG VACUUM_PUMP_C (Follower) 2aef81a6-135c-4ce5-9b47-01576e635930 added.
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy "MyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 88354d45-5265-4ed1-b51a-acd9f9f77f15":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "ROTARY_FEEDER_C (Leader)": Instantiated

```

(continues on next page)

(continued from previous page)

```

YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "ROTARY_FEEDER_C (Leader) 88354d45-5265-
↳4ed1-b51a-acd9f9f77f15": Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 88354d45-5265-4ed1-b51a-acd9f9f77f15":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "ROTARY_FEEDER_C (Leader) 88354d45-5265-
↳4ed1-b51a-acd9f9f77f15": Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 88354d45-5265-4ed1-b51a-acd9f9f77f15":
↳Adaptivity switched on
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
↳GT Player SG ROTARY_FEEDER_C (Leader) 88354d45-5265-4ed1-b51a-acd9f9f77f15 added.
YYYY-MM-DD HH:MM:SS.SSSSSS I Training "GT Training": Training started (without
↳hyperparameter tuning)
YYYY-MM-DD HH:MM:SS.SSSSSS I Results "RL": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -- Training run 0 started...
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳-----

YYYY-MM-DD HH:MM:SS.SSSSSS I GT Game "Matrix": Process time 0:00:00 : Scenario reset
↳with seed 0
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Reset
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳-----
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -- Training episode 0 started...
YYYY-MM-DD HH:MM:SS.SSSSSS W Training "GT Training": -----
↳-----

YYYY-MM-DD HH:MM:SS.SSSSSS S GT Game "Matrix": Process time 0:00:00 : Start of cycle 0
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Game "Matrix": Process time 0:00:00 : Agent computes
↳action...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
↳Start of action computation for all agents...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "BELT_CONVEYOR_A (Leader) 478c892f-f980-
↳4f5a-b894-f6d57dd357f7": Action computation started
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "BELT_CONVEYOR_A (Leader) 478c892f-f980-
↳4f5a-b894-f6d57dd357f7": Action computation finished
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "ROTARY_FEEDER_C (Leader) 88354d45-5265-
↳4ed1-b51a-acd9f9f77f15": Action computation started
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "ROTARY_FEEDER_C (Leader) 88354d45-5265-
↳4ed1-b51a-acd9f9f77f15": Action computation finished
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "VACUUM_PUMP_B (Follower) 61ceca2b-52fc-
↳4f34-8fbb-36bff56d9e1e": Action computation started
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "VACUUM_PUMP_B (Follower) 61ceca2b-52fc-
↳4f34-8fbb-36bff56d9e1e": Action computation finished
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "VIBRATORY_CONVEYOR_B (Follower) 6b4f1ea8-
↳9b24-48b1-b4bc-f58662dc9cef": Action computation started
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "VIBRATORY_CONVEYOR_B (Follower) 6b4f1ea8-

```

(continues on next page)

(continued from previous page)

```

→9b24-48b1-b4bc-f58662dc9cef": Action computation finished
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "VACUUM_PUMP_C (Follower) 2aef81a6-135c-
→4ce5-9b47-01576e635930": Action computation started
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player SG "VACUUM_PUMP_C (Follower) 2aef81a6-135c-
→4ce5-9b47-01576e635930": Action computation finished
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
→End of action computation for all agents...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Game "Matrix": Process time 0:00:00 : Env processes
→action...
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Start processing action
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Actions of agent 478c892f-f980-
→4f5a-b894-f6d57dd357f7 = [0.84442185]
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Actions of agent 88354d45-5265-
→4ed1-b51a-acd9f9f77f15 = [0.7579544]
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Actions of agent 61ceca2b-52fc-
→4f34-8fbb-36bff56d9e1e = [0.42057158]
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Actions of agent 6b4f1ea8-9b24-
→48b1-b4bc-f58662dc9cef = [0.25891675]
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Actions of agent 2aef81a6-135c-
→4ce5-9b47-01576e635930 = [0.51127472]
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Assessment for success...
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Assessment for breakdown...
YYYY-MM-DD HH:MM:SS.SSSSSS I Game Board "BGLP_GT": Action processing finished
→successfully
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Game "Matrix": Process time 0:00:01 : Agent adapts
→policy...
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Multi-Player SG "BGLP Players with Random Policies":
→Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
→Start of adaptation for all agents...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
→Start adaption for agent 478c892f-f980-4f5a-b894-f6d57dd357f7
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "BELT_CONVEYOR_A (Leader) 478c892f-f980-
→4f5a-b894-f6d57dd357f7": Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 478c892f-f980-4f5a-b894-f6d57dd357f7":
→Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy "MyPolicy 478c892f-f980-4f5a-b894-f6d57dd357f7":
→Sorry, I am a stupid agent...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
→Start adaption for agent 88354d45-5265-4ed1-b51a-acd9f9f77f15
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "ROTARY_FEEDER_C (Leader) 88354d45-5265-
→4ed1-b51a-acd9f9f77f15": Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 88354d45-5265-4ed1-b51a-acd9f9f77f15":
→Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy "MyPolicy 88354d45-5265-4ed1-b51a-acd9f9f77f15":
→Sorry, I am a stupid agent...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
→Start adaption for agent 61ceca2b-52fc-4f34-8fbb-36bff56d9e1e
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "VACUUM_PUMP_B (Follower) 61ceca2b-52fc-
→4f34-8fbb-36bff56d9e1e": Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 61ceca2b-52fc-4f34-8fbb-36bff56d9e1e":
→Adaptation started

```

(continues on next page)

(continued from previous page)

```

YYYY-MM-DD HH:MM:SS.SSSSSS I Policy "MyPolicy 61ceca2b-52fc-4f34-8fbb-36bff56d9e1e":
↳ Sorry, I am a stupid agent...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
↳ Start adaption for agent 6b4f1ea8-9b24-48b1-b4bc-f58662dc9cef
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "VIBRATORY_CONVEYOR_B (Follower) 6b4f1ea8-
↳ 9b24-48b1-b4bc-f58662dc9cef": Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 6b4f1ea8-9b24-48b1-b4bc-f58662dc9cef":
↳ Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy "MyPolicy 6b4f1ea8-9b24-48b1-b4bc-f58662dc9cef":
↳ Sorry, I am a stupid agent...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
↳ Start adaption for agent 2aef81a6-135c-4ce5-9b47-01576e635930
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Player SG "VACUUM_PUMP_C (Follower) 2aef81a6-135c-
↳ 4ce5-9b47-01576e635930": Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS S Policy "MyPolicy 2aef81a6-135c-4ce5-9b47-01576e635930":
↳ Adaptation started
YYYY-MM-DD HH:MM:SS.SSSSSS I Policy "MyPolicy 2aef81a6-135c-4ce5-9b47-01576e635930":
↳ Sorry, I am a stupid agent...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Multi-Player SG "BGLP Players with Random Policies":
↳ End of adaptation for all agents...
YYYY-MM-DD HH:MM:SS.SSSSSS S GT Game "Matrix": Process time 0:00:01 : End of cycle 0

....

YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Training Results of run 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Scenario : GT Game Matrix
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Model : GT Multi-Player SG
↳ BGLP Players with Random Policies
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Start time stamp : YYYY-MM-DD
↳ HH:MM:SS.SSSSSS
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- End time stamp : YYYY-MM-DD
↳ HH:MM:SS.SSSSSS
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Duration : 0:00:02.156790
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Start cycle id : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- End cycle id : 199
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Training cycles : 200
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Evaluation cycles : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Adaptations : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- High score : None
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Training Episodes : 2
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -- Evaluations : 0
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -----
↳ -----
YYYY-MM-DD HH:MM:SS.SSSSSS W Results "RL": -----
↳ -----

```

(continues on next page)

(continued from previous page)

```
YYYY-MM-DD HH:MM:SS.SSSSS I Training "GT Training": Training completed
```

## Cross Reference

- *API Reference: Game Theory in Dynamic Games*

## 10.4.2 Native GT

### Howto GT-Native-001: 2P Prisoners' Dilemma

#### Executable code

```
## -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.gt.examples
## -- Module  : howto_gt_native_001_prisoners_dilemma_2p.py
## -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-09-22  0.0.0     SY       Creation
## -- 2023-12-12  1.0.0     SY       Release of first version
## -- 2024-01-05  1.0.1     SY       Renaming
## -----
"""
Ver. 1.0.1 (2024-01-05)

This module shows how to run a game, namely 2P Prisoners' Dilemma.

You will learn:

1) How to set up a game, including solver, competition, coalition, payoff, and more
2) How to run the game
3) How to analyse the game
"""

from mlpro.gt.native.basics import *
from mlpro.gt.pool.native.games.prisonersdilemma_2p import *
from pathlib import Path

if __name__ == "__main__":
    cycle_limit = 10
    logging      = Log.C_LOG_ALL
    visualize    = False
```

(continues on next page)

(continued from previous page)

```

path      = str(Path.home())

else:
    cycle_limit = 1
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    path        = None

training = GTTraining(
    p_game_cls=PrisonersDilemma2PGame,
    p_cycle_limit=cycle_limit,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging
)

training.run()

```

## Results

```

YYYY-MM-DD HH:MM:SS.SSSSSS I GT Training "Native GT Training": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Game "PrisonersDilemma2PGame": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "RandomSolver": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player of Prisoner 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player of Prisoner 1": Player of Prisoner 1
→ is keeping the same solver 1
...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Training "Native GT Training": Training completed

```

## Cross Reference

- [API Reference: Native Game Theory](#)

## Howto GT-Native-002: 3P Prisoners' Dilemma

### Executable code

```

## -----
→ -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.gt.examples
## -- Module  : howto_gt_native_002_prisoners_dilemma_3p.py
## -----
→ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2023-12-07  0.0.0     SY         Creation
## -- 2023-12-12  1.0.0     SY         Release of first version
## -- 2024-01-05  1.0.1     SY         Renaming
## -----
→ -----

```

(continues on next page)



(continued from previous page)

```

"""
Ver. 1.0.1 (2024-01-05)

This module shows how to run a game, namely 3P Prisoners' Dilemma with two solvers, such
↳as random
solver and min greedy policy.

You will learn:

1) How to set up a game, including solver, competition, coalition, payoff, and more
2) How to run the game
3) How to analyse the game

"""

from mlpro.gt.native.basics import *
from mlpro.gt.pool.native.games.prisonersdilemma_3p import *
from pathlib import Path

if __name__ == "__main__":
    cycle_limit = 10
    logging      = Log.C_LOG_ALL
    visualize    = False
    path         = str(Path.home())

else:
    cycle_limit = 1
    logging      = Log.C_LOG_NOTHING
    visualize    = False
    path         = None

training = GTTraining(
    p_game_cls=PrisonersDilemma3PGame,
    p_cycle_limit=cycle_limit,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging
)

training.run()

```

## Results

```

YYYY-MM-DD HH:MM:SS.SSSSSS I GT Training "Native GT Training": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Game "PrisonersDilemma3PGame": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "RandomSolver": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "MinGreedyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player of Prisoner 1": Instantiated

```

(continues on next page)



(continued from previous page)

```

YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player of Prisoner 1": Player 1 is switching.
↳to solver MinGreedyPolicy 1
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Prisoner 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Prisoner 1": Player of
↳Prisoner 1 added.
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "RandomSolver": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "MinGreedyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player of Prisoner 2": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player of Prisoner 2": Player 2 is switching.
↳to solver RandomSolver 2
...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Training "Native GT Training": Training completed

```

## Cross Reference

- *API Reference: Native Game Theory*

## Howto GT-Native-003: Rock, Paper, Scissors

### Executable code

```

## -----
↳-----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.gt.examples
## -- Module  : howto_gt_native_003_rock_paper_scissors.py
## -----
↳-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-12-08  0.0.0     SY       Creation
## -- 2023-12-12  1.0.0     SY       Release of first version
## -----
↳-----

"""
Ver. 1.0.0 (2023-12-12)

This module shows how to run a duel of two coalitions of a game of Rock Paper Scissors.
You will learn:

1) How to set up a game, including solver, competition, coalition, payoff, and more
2) How to run the game
3) How to analyse the game

"""

from mlpro.gt.native.basics import *
from mlpro.gt.pool.native.games.rockpaperscissors import *

```

(continues on next page)

(continued from previous page)

```

from pathlib import Path

if __name__ == "__main__":
    cycle_limit = 10
    logging      = Log.C_LOG_ALL
    visualize    = False
    path         = str(Path.home())

else:
    cycle_limit = 1
    logging      = Log.C_LOG_NOTHING
    visualize    = False
    path         = None

training = GTTraining(
    p_game_cls=RockPaperScissors,
    p_cycle_limit=cycle_limit,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging
)

training.run()

```

## Results

```

YYYY-MM-DD HH:MM:SS.SSSSSS I GT Training "Native GT Training": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Game "RockPaperScissors": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "RandomSolver": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 1 of Team 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 1 of Team 1": Player 1 of Team 1 is_
↳keeping the same solver 1
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 2 of Team 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 2 of Team 1": Player 2 of Team 1 is_
↳keeping the same solver 1
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 3 of Team 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 3 of Team 1": Player 3 of Team 1 is_
↳keeping the same solver 1
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 4 of Team 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 4 of Team 1": Player 4 of Team 1 is_
↳keeping the same solver 1
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 5 of Team 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 5 of Team 1": Player 5 of Team 1 is_
↳keeping the same solver 1
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Team 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Team 1": Player 1 of Team 1_
↳added.
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Team 1": Player 2 of Team 1_
↳added.
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Team 1": Player 3 of Team 1_

```

(continues on next page)

(continued from previous page)

```

↪added.
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Team 1": Player 4 of Team 1↪
↪added.
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Team 1": Player 5 of Team 1↪
↪added.
...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Training "Native GT Training": Training completed

```

## Cross Reference

- *API Reference: Native Game Theory*

## Howto GT-Native-004: 3P Supply and Demand

### Executable code

```

## -----
↪-----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.gt.examples
## -- Module  : howto_gt_native_004_supply_demand_3p.py
## -----
↪-----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.      Description
## -- 2023-12-12  0.0.0     SY         Creation
## -- 2023-12-12  1.0.0     SY         Release of first version
## -- 2024-01-12  1.0.1     SY         Refactoring
## -----
↪-----

"""
Ver. 1.0.1 (2024-01-12)

This module shows how to run a 3 sellers competition game of supply and demand.

You will learn:

1) How to set up a game, including solver, competition, coalition, payoff, and more
2) How to run the game
3) How to analyse the game

"""

from mlpro.gt.native.basics import *
from mlpro.gt.pool.native.games.supplydemand_3p import *
from pathlib import Path

```

(continues on next page)

(continued from previous page)

```

if __name__ == "__main__":
    cycle_limit = 10
    logging      = Log.C_LOG_ALL
    visualize    = False
    path         = str(Path.home())

else:
    cycle_limit = 1
    logging      = Log.C_LOG_NOTHING
    visualize    = False
    path         = None

training = GTTraining(
    p_game_cls=SupplyDemand_3P,
    p_cycle_limit=cycle_limit,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging
)

training.run()

```

## Results

```

YYYY-MM-DD HH:MM:SS.SSSSSS I GT Game "SupplyDemand_3P": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "MaxGreedyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Seller 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Seller 1": Seller 1 is keeping the same.
↪ solver 0
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Seller 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Seller 1": Seller 1 added.
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "MaxGreedyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Seller 2": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Seller 2": Seller 2 is keeping the same.
↪ solver 1
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Seller 2": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Seller 2": Seller 2 added.
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "RandomSolver": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Seller 3": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Seller 3": Seller 3 is keeping the same.
↪ solver 2
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Seller 3": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Seller 3": Seller 3 added.
...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Training "Native GT Training": Training completed

```

## Cross Reference

- [API Reference: Native Game Theory](#)

## Howto GT-Native-005: 3P Routing Problems

### Executable code

```
## -----
↪ -----
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.gt.examples
## -- Module  : howto_gt_native_005_routing_problems_3p.py
## -----
↪ -----
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2024-01-12  0.0.0     SY       Creation
## -- 2024-01-12  1.0.0     SY       Release of first version
## -----
↪ -----

"""
Ver. 1.0.0 (2024-01-12)

This module shows how to run a 3P competition game of routing problems.

You will learn:

1) How to set up a game, including solver, competition, coalition, payoff, and more
2) How to run the game
3) How to analyse the game

"""

from mlpro.gt.native.basics import *
from mlpro.gt.pool.native.games.routingproblems_3p import *
from pathlib import Path

if __name__ == "__main__":
    cycle_limit = 10
    logging      = Log.C_LOG_ALL
    visualize    = False
    path         = str(Path.home())
else:
    cycle_limit = 1
    logging      = Log.C_LOG_NOTHING
    visualize    = False
    path         = None

training = GTTraining(
    p_game_cls=Routing_3P,
```

(continues on next page)

(continued from previous page)

```

        p_cycle_limit=cycle_limit,
        p_path=path,
        p_visualize=visualize,
        p_logging=logging
    )

training.run()

```

## Results

```

YYYY-MM-DD HH:MM:SS.SSSSSS I GT Training "Native GT Training": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Game "Routing_3P": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "MinGreedyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 1": Player 1 is keeping the same.
↪ solver 0
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Player 1": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Player 1": Player 1 added.
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "MinGreedyPolicy": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "RandomSolver": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 2": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 2": Player 1 is switching to solver.
↪ MinGreedyPolicy 1
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Player 2": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Coalition "Coalition of Player 2": Player 2 added.
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Solver "RandomSolver": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 3": Instantiated
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Player "Player 3": Player 3 is keeping the same.
↪ solver 2
...
YYYY-MM-DD HH:MM:SS.SSSSSS I GT Training "Native GT Training": Training completed

```

## Cross Reference

- [API Reference: Native Game Theory](#)

## 10.5 MLPro-OA - Online Adaptivity

Coming soon...

## BF-VARIOUS - Various Functions



- **p\_id** – Optional external id
- **Attributes**
- **\_id** – Unique id of the object.

**get\_id()**

**set\_id(p\_id=None)**

Sets/generates a new id.

**Parameters**

**p\_id** – Optional external id. If None, a unique id is generated.

**class mlpro.bf.various.Log(p\_logging=True)**

Bases: object

This class adds elementary log functionality to inherited classes.

**Parameters**

**p\_logging** – Log level (see constants C\_LOG\_\*). Default: Log.C\_LOG\_ALL

**C\_TYPE** = '????'

**C\_NAME** = '????'

**C\_LOG\_TYPE\_I** = 'I'

**C\_LOG\_TYPE\_W** = 'W'

**C\_LOG\_TYPE\_E** = 'E'

**C\_LOG\_TYPE\_S** = 'S'

**C\_LOG\_TYPES** = ['I', 'W', 'E', 'S']

**C\_COL\_WARNING** = '\x1b[93m'

**C\_COL\_ERROR** = '\x1b[91m'

**C\_COL\_SUCCESS** = '\x1b[32m'

**C\_COL\_RESET** = '\x1b[0m'

**C\_LOG\_ALL** = True

**C\_LOG\_NOTHING** = False

**C\_LOG\_WE** = 'W'

**C\_LOG\_E** = 'E'

**C\_LOG\_LEVELS** = [True, False, 'W', 'E']

**C\_INST\_MSG** = True

**get\_name()** → str

**set\_name(p\_name: str)**



**switch\_logging**(*p\_logging*)

Sets new log level.

**Parameters**

**p\_logging** – Log level (constant C\_LOG\_LEVELS contains valid values)

**get\_log\_level**()

**log**(*p\_type*, \**p\_args*)

Writes log line to standard output in format: yyyy-mm-dd hh:mm:ss.mmmmmm [*p\_type* C\_TYPE C\_NAME]: [*p\_args*]

**Parameters**

- **entry** (*p\_type* type of log)
- **informations** (*p\_args* log)

**Returns**

Nothing

**class** mlpro.bf.various.**Persistent**(*p\_id=None*, *p\_logging=True*)

Bases: [Id](#), [Log](#)

Property class that inherits persistence to its child classes.

**Parameters**

- **p\_id** – Optional external id
- **p\_logging** – Log level (see constants C\_LOG\_\*). Default: Log.C\_LOG\_ALL

**C\_PERSISTENCE\_VERSION**

Version of the implementation of the persistence. Shall be raised in child classes whenever an incompatible change has been done.

**Type**

str

**C\_SUFFIX**

Default suffix for pickled result files.

**Type**

str = '.pkl'

**C\_PERSISTENCE\_VERSION: str = '1.0.0'**

**C\_SUFFIX: str = '.pkl'**

**get\_filename\_stub**() → str

Returns the unique filename of the object without a suffix.

**Returns**

**filename\_stub** – Filename stub.

**Return type**

str

**get\_filename**() → str

Returns the full unique filename of the object including the suffix.

**Returns**

**filename** – Full filename.

**Return type**

str

**set\_filename**(*p\_filename\_stub*: str, *p\_suffix*: str = None)**\_get\_path**() → str

Internal helper method to determine the current path for loading/saving external data.

**classmethod load**(*p\_path*: str, *p\_filename*: str)

Static method to load an object of the current class from a file using pickle/dill. During unpickling the given file, standard method `__setstate__()` is called. This in turn is implemented specifically and calls the MLPro custom method `_complete_state()`. This method allows the completion of the unpickled object from further externally stored data.

**Parameters**

- **p\_path** (str) – Path where file will be saved
- **p\_filename** (str = None) – File name (if None an internal filename will be used)

**Returns**

Object of the given class that was unpickled from the given file.

**Return type**

Object

**\_complete\_state**(*p\_path*: str, *p\_os\_sep*: str, *p\_filename\_stub*: str)

Custom method to complete the object state (=self) from external data sources. This method is called by standard method `__setstate__()` during unpickling the object from an external file.

**Parameters**

- **p\_path** (str) – Path of the object pickle file (and further optional related files)
- **p\_os\_sep** (str) – OS-specific path separator.
- **p\_filename\_stub** (str) – Filename stub to be used for further optional custom data files

**save**(*p\_path*: str, *p\_filename*: str = None) → bool

Saves the object to the given path and file name using pickle/dill. If file name is None, a unique internal file name is used (recommended). During pickling the Python standard method `__getstate__()` is called. This in turn is implemented specifically and calls the MLPro custom method `_reduce_state()`. This method allows to reduce unpickleable components from the object state before pickling. These components can optionally be stored in separate files of a suitable format.

**Parameters**

- **p\_path** (str) – Path where file will be saved
- **p\_filename** (str = None) – File name (if None an internal filename will be used)

**Returns****successful** – True, if file content was saved successfully. False otherwise.**Return type**

bool

**\_reduce\_state**(*p\_state*: dict, *p\_path*: str, *p\_os\_sep*: str, *p\_filename\_stub*: str)

Custom method to reduce the given object state by components that can not be pickled. Further data files can be created in the given path and should use the given filename stub.

**Parameters**

- **p\_state** (*dict*) – Object state dictionary to be reduced by components that can not be pickled.
- **p\_path** (*str*) – Path to store further optional custom data files
- **p\_os\_sep** (*str*) – OS-specific path separator.
- **p\_filename\_stub** (*str*) – Filename stub to be used for further optional custom data files

**class** mlpro.bf.various.**Timer**(*p\_mode: int, p\_lap\_duration: timedelta = None, p\_lap\_limit: int = 999999*)

Bases: object

Timer class in two time modes (real/virtual) and with simple lap management.

#### Parameters

- **p\_mode** (*int*) – C\_MODE\_REAL for real time mode or C\_MODE\_VIRTUAL for virtual time mode
- **p\_lap\_duration** (*timedelta = None*) – Optional duration of a single lap.
- **p\_lap\_limit** (*int = C\_LAP\_LIMIT*) – Maximum number of laps until the lap counter restarts with 0

**C\_MODE\_REAL** = 0

**C\_MODE\_VIRTUAL** = 1

**C\_LAP\_LIMIT** = 999999

**reset()** → None

Resets timer.

#### Returns

Nothing

**get\_time()** → timedelta

**get\_lap\_time()** → timedelta

**get\_lap\_id()**

**add\_time**(*p\_delta: timedelta*)

**finish\_lap()** → bool

Finishes the current lap. In timer mode C\_MODE\_REAL the remaining time until the end of the lap will be paused.

#### Returns

True, if the remaining time to the next lap was positive. False, if the timer timed out.

**class** mlpro.bf.various.**TStamp**(*p\_tstamp: timedelta = None*)

Bases: object

This class provides elementary time stamp functionality for inherited classes.

**get\_tstamp()** → timedelta

**set\_tstamp**(*p\_tstamp: timedelta*)

```
class mlpro.bf.various.ScientificObject
```

```
    Bases: object
```

```
    This class provides elementary functionality for storing a scientific reference.
```

```
    C_SCIREF_TYPE_NONE = None
```

```
    C_SCIREF_TYPE_ARTICLE = 'Article'
```

```
    C_SCIREF_TYPE_BOOK = 'Book'
```

```
    C_SCIREF_TYPE_BOOKLET = 'Booklet'
```

```
    C_SCIREF_TYPE_INBOOK = 'Inbook'
```

```
    C_SCIREF_TYPE_ONLINE = 'Online'
```

```
    C_SCIREF_TYPE_PROCEEDINGS = 'Proceedings'
```

```
    C_SCIREF_TYPE_INPROCEEDINGS = 'Inproceedings'
```

```
    C_SCIREF_TYPE_TECHREPORT = 'Technical Report'
```

```
    C_SCIREF_TYPE_UNPUBLISHED = 'Unpublished'
```

```
    C_SCIREF_TYPE = None
```

```
    C_SCIREF_AUTHOR = None
```

```
    C_SCIREF_TITLE = None
```

```
    C_SCIREF_JOURNAL = None
```

```
    C_SCIREF_ABSTRACT = None
```

```
    C_SCIREF_VOLUME = None
```

```
    C_SCIREF_NUMBER = None
```

```
    C_SCIREF_PAGES = None
```

```
    C_SCIREF_YEAR = None
```

```
    C_SCIREF_MONTH = None
```

```
    C_SCIREF_DAY = None
```

```
    C_SCIREF_DOI = None
```

```
    C_SCIREF_KEYWORDS = None
```

```
    C_SCIREF_ISBN = None
```

```
    C_SCIREF_SERIES = None
```

```
    C_SCIREF_PUBLISHER = None
```

```
    C_SCIREF_CITY = None
```

```
    C_SCIREF_COUNTRY = None
```

```

C_SCIREF_URL = None
C_SCIREF_CHAPTER = None
C_SCIREF_BOOKTITLE = None
C_SCIREF_INSTITUTION = None
C_SCIREF_CONFERENCE = None
C_SCIREF_NOTES = None
C_SCIREF_EDITOR = None
C_SCIREF_ADDRESS = None
C_SCIREF_HOWPUBLISHED = None
C_SCIREF_NUMPAGES = None
C_SCIREF_ISSN = None
C_SCIREF_VERSION = None

```

```
get_bibtex()
```

```
class mlpro.bf.various.PersonalisedStamp(p_name: str, p_id: int = None)
```

Bases: [Id](#)

This class serves as a base class of label to set up a name and id for another class.

#### Parameters

- **p\_name** (*str*) – name of the created class.
- **p\_id** (*int*) – unique id of the created class. Default: None.

#### C\_NAME

name of the created class. Default: ‘’.

#### Type

str

```
C_NAME = ''
```

```
set_name(p_name: str)
```

This method provides a functionality to set an unique name.

#### Parameters

**p\_name** (*str*) – An unique name.

```
get_name() → str
```

This method provides a functionality to get the unique name.

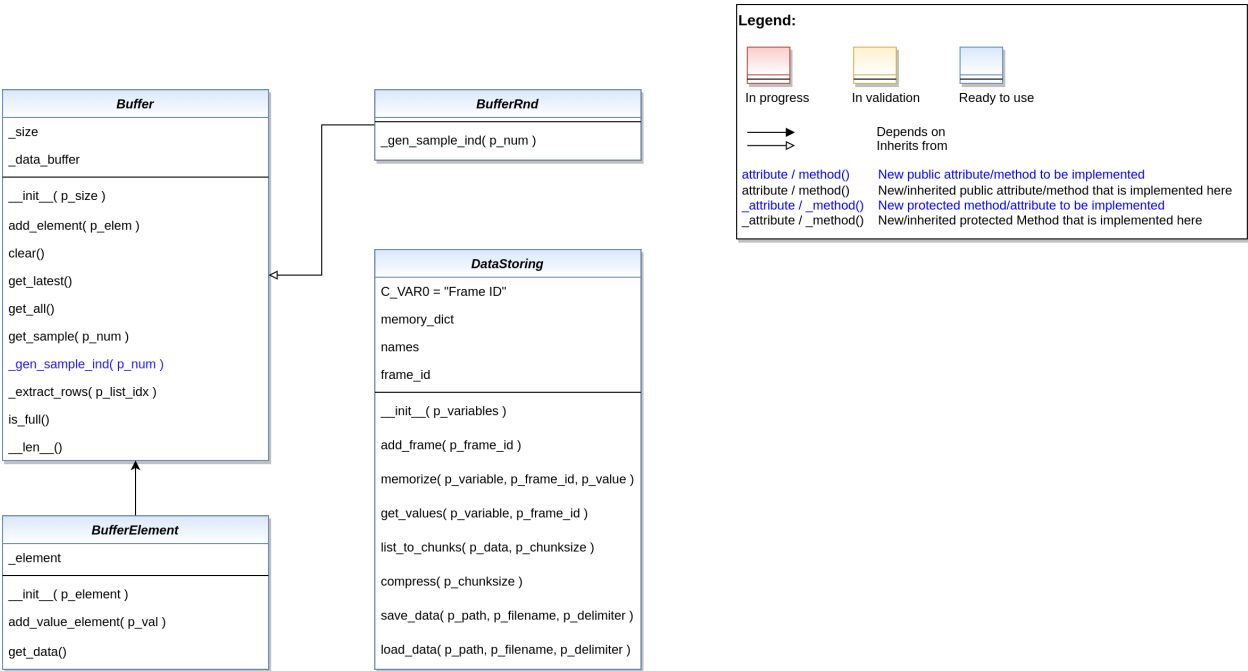
#### Returns

The unique name of the related component.

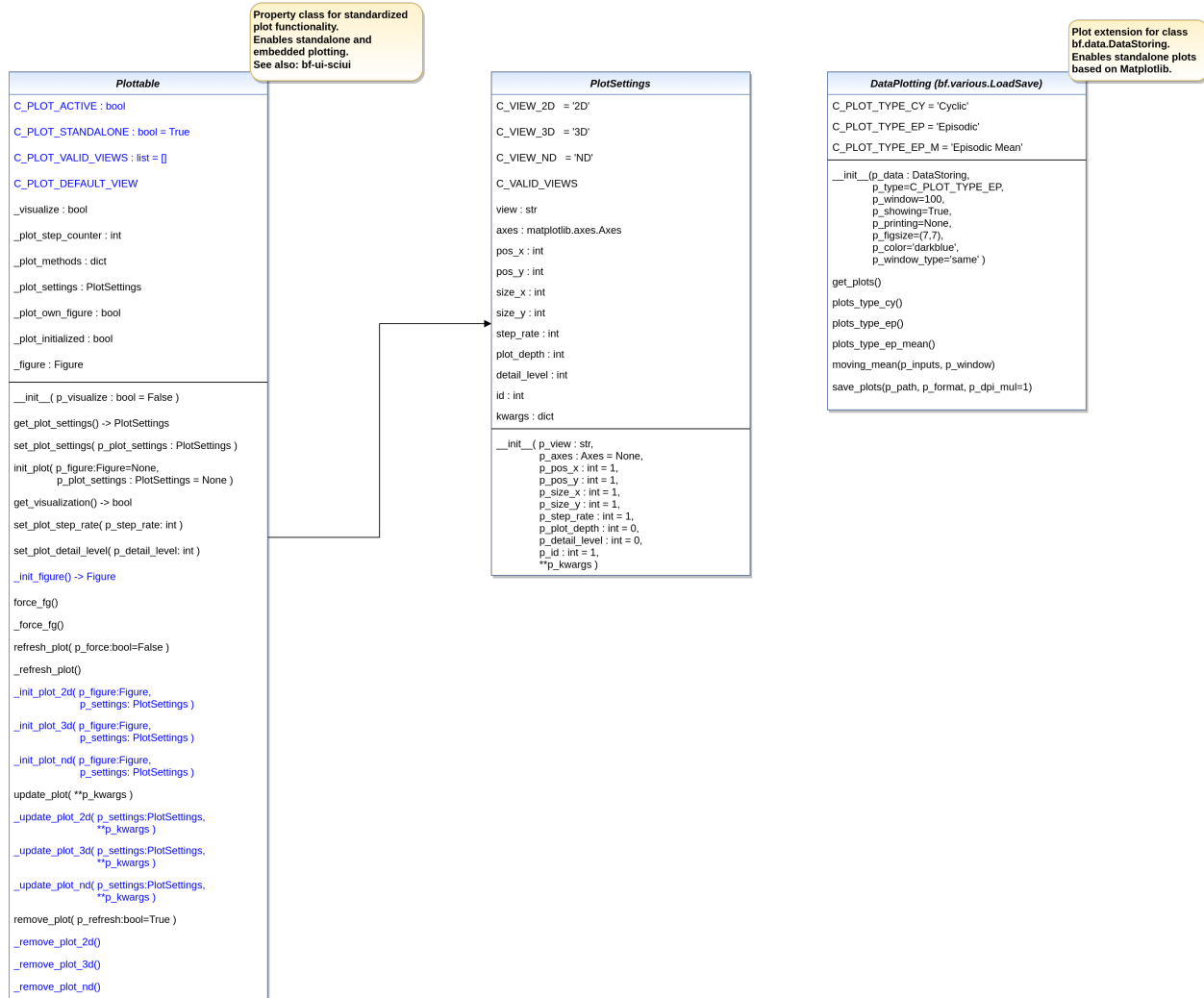
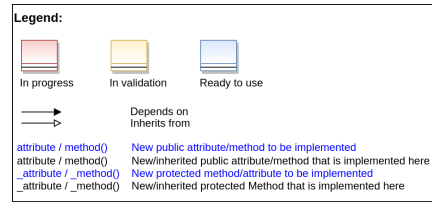
#### Return type

str

BF-DATA - Data Management



## BF-PLOT - Plotting and Visualization



Ver. 2.11.1 (2024-02-24)

This module provides various classes related to data plotting.

**class** mlpro.bf.plot.**PlotSettings**(*p\_view*: str, *p\_axes*: Axes = None, *p\_pos\_x*: int = 1, *p\_pos\_y*: int = 1, *p\_size\_x*: int = 1, *p\_size\_y*: int = 1, *p\_step\_rate*: int = 1, *p\_horizon*: int = 0, *p\_plot\_depth*: int = 0, *p\_detail\_level*: int = 0, *p\_force\_fg*: bool = True, *p\_id*: int = 1, *p\_view\_autoselect*: bool = True, *\*\*p\_kwargs*)

Bases: object

Class to specify the context of a subplot.

**Parameters**

- **p\_view** (*str*) – ID of the view (see constants C\_VIEW\_\*)
- **p\_axes** (*Axes*) – Optional Matplotlib Axes object as destination for plot outputs. Default is None.
- **p\_pos\_x** (*int*) – Optional x position of a subplot within a Matplotlib figure. Default = 1.
- **p\_pos\_y** (*int*) – Optional y position of a subplot within a Matplotlib figure. Default = 1.
- **p\_size\_x** (*int*) – Relative size factor in x direction. Default = 1.
- **p\_size\_y** (*int*) – Relative size factor in y direction. Default = 1.
- **p\_step\_rate** (*int*) – Optional step rate. Decides after how many calls of the update\_plot() method the custom methods \_update\_plot() carries out an output. Default = 1.
- **p\_horizon** (*int*) – Optional plot horizon. A value > 0 limits the number of data entities that are shown in the plot. Default = 0.
- **p\_plot\_depth** (*int*) – Optional plot depth in case of hierarchical plotting. A value of 0 means that the plot depth is unlimited. Default = 0.
- **p\_detail\_level** (*int*) – Optional detail level. Default = 0.
- **p\_force\_fg** (*bool*) – Optional boolean flag. If True, the releated window is kept in foreground. Default = True.
- **p\_id** (*int*) – Optional unique id of the subplot within the figure. Default = 1.
- **p\_view\_autoselect** (*bool*) – If True, the final view is automatically selected during runtime. Default = True.
- **p\_kwargs** (*dict*) – Further optional named parameters.

C\_VIEW\_2D = '2D'

C\_VIEW\_3D = '3D'

C\_VIEW\_ND = 'ND'

C\_VALID\_VIEWS = ['2D', '3D', 'ND']

**class** mlpro.bf.plot.Plottable(*p\_visualize: bool = False*)

Bases: object

Property class that inherits the ability to be plottable. The class is prepared for plotting with MatPlotLib but not restricted to it. Three different views are supported:

2D: 2-dimensional plot 3D: 3-dimensional plot ND: Multidimensional plot

See class Plotsettings for further details.

**Parameters**

- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.

**C\_PLOT\_ACTIVE**

Custom attribute to turn on or off the plot functionality. Must be turned on explicitly.

**Type**

bool



**C\_PLOT\_STANDALONE**

Custom attribute to be set to True, if the plot needs a separate subplot or False if the plot can be added to an existing subplot.

**Type**

bool = True

**C\_PLOT\_VALID\_VIEWS**

Custom list of views that are supported/implemented (see class PlotSettings)

**Type**

list = [[PlotSettings](#)]

**C\_PLOT\_DEFAULT\_VIEW**

Custom attribute for the default view. See class PlotSettings for more details.

**Type**

str = ''

**C\_PLOT\_ACTIVE:** bool = False

**C\_PLOT\_STANDALONE:** bool = True

**C\_PLOT\_VALID\_VIEWS:** list = []

**C\_PLOT\_DEFAULT\_VIEW:** str = 'ND'

**get\_plot\_settings()** → [PlotSettings](#)

**set\_plot\_settings**(*p\_plot\_settings*: [PlotSettings](#))

Sets plot settings in advance (before initialization of plot).

**Parameters**

**p\_plot\_settings** ([PlotSettings](#)) – New PlotSettings to be set. If None, the default view is plotted (see attribute C\_PLOT\_DEFAULT\_VIEW).

**init\_plot**(*p\_figure*: *Figure = None*, *p\_plot\_settings*: [PlotSettings = None](#), *\*\*p\_kwargs*)

Initializes the plot functionalities of the class.

**Parameters**

- **p\_figure** (*Matplotlib.figure.Figure*, *optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p\_plot\_settings** ([PlotSettings](#)) – Optional plot settings. If None, the default view is plotted (see attribute C\_PLOT\_DEFAULT\_VIEW).

**get\_visualization()** → bool

**set\_plot\_step\_rate**(*p\_step\_rate*: int)

**set\_plot\_detail\_level**(*p\_detail\_level*: int)

**\_init\_figure()** → *Figure*

Custom method to initialize a suitable standalone Matplotlib figure.

**Returns**

**figure** – Matplotlib figure object to host the subplot(s)

**Return type**

Matplotlib.figure.Figure

**force\_fg()**

Internal use.

**\_force\_fg**(*p\_fig: Figure*)

Internal use.

**refresh\_plot**(*p\_force: bool = False*)

Refreshes the plot.

**Parameters**

**p\_force** (*bool = False*) – On True the plot is updated even if it is embedded in a foreign host figure.

**\_refresh\_plot()**

Custom method to refresh the plot. Default implementation assumes standard use of Matplotlib.

**\_init\_plot\_2d**(*p\_figure: Figure, p\_settings: PlotSettings*)

Custom method to initialize a 2D plot. If attribute `p_settings.axes` is not None the initialization shall be done there. Otherwise a new MatPlotLib Axes object shall be created in the given figure and stored in `p_settings.axes`.

Note: Please call this method in your custom implementation to create a default subplot.

**Parameters**

- **p\_figure** (*Matplotlib.figure.Figure*) – Matplotlib figure object to host the subplot(s).
- **p\_settings** (*PlotSettings*) – Object with further plot settings.

**\_init\_plot\_3d**(*p\_figure: Figure, p\_settings: PlotSettings*)

Custom method to initialize a 3D plot. If attribute `p_settings.axes` is not None the initialization shall be done there. Otherwise a new MatPlotLib Axes object shall be created in the given figure and stored in `p_settings.axes`.

Note: Please call this method in your custom implementation to create a default subplot.

**Parameters**

- **p\_figure** (*Matplotlib.figure.Figure*) – Matplotlib figure object to host the subplot(s).
- **p\_settings** (*PlotSettings*) – Object with further plot settings.

**\_init\_plot\_nd**(*p\_figure: Figure, p\_settings: PlotSettings*)

Custom method to initialize a nD plot. If attribute `p_settings.axes` is not None the initialization shall be done there. Otherwise a new MatPlotLib Axes object shall be created in the given figure and stored in `p_settings.axes`.

Note: Please call this method in your custom implementation to create a default subplot.

**Parameters**

- **p\_figure** (*Matplotlib.figure.Figure*) – Matplotlib figure object to host the subplot(s).
- **p\_settings** (*PlotSettings*) – Object with further plot settings.

**update\_plot**(*\*\*p\_kwargs*)

Updates the plot.

**Parameters**

**\*\*p\_kwargs** – Implementation-specific plot data and/or parameters.

**\_update\_plot\_2d**(*p\_settings*: [PlotSettings](#), *\*\*p\_kwargs*)

Custom method to update the 2d plot. The related Matplotlib Axes object is stored in *p\_settings*.

**Parameters**

- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.
- **\*\*p\_kwargs** – Implementation-specific data and parameters.

**\_update\_plot\_3d**(*p\_settings*: [PlotSettings](#), *\*\*p\_kwargs*)

Custom method to update the 3d plot. The related Matplotlib Axes object is stored in *p\_settings*.

**Parameters**

- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.
- **\*\*p\_kwargs** – Implementation-specific data and parameters.

**\_update\_plot\_nd**(*p\_settings*: [PlotSettings](#), *\*\*p\_kwargs*)

Custom method to update the nd plot. The related Matplotlib Axes object is stored in *p\_settings*.

**Parameters**

- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.
- **\*\*p\_kwargs** – Implementation-specific data and parameters.

**remove\_plot**(*p\_refresh*: *bool = True*)

” Removes the plot and optionally refreshes the display.

**Parameters**

**p\_refresh** (*bool = True*) – On True the display is refreshed after removal

**\_remove\_plot\_2d**()

Custom method to remove 2D plot artifacts when object is destroyed.

**\_remove\_plot\_3d**()

Custom method to remove 3D plot artifacts when object is destroyed.

**\_remove\_plot\_nd**()

Custom method to remove nd plot artifacts when object is destroyed.

```
class mlpro.bf.plot.DataPlotting(p_data: DataStoring, p_type='Episodic', p_window=100,
                                p_showing=True, p_printing=None, p_figsize=(7, 7), p_color='darkblue',
                                p_window_type='same')
```

Bases: [Persistent](#)

This class provides a functionality to plot the stored values of variables.

**Parameters**

- **p\_data** (*DataStoring*) – Data object with stored variables values.
- **p\_type** (*str, optional*) – Type of plot. The default is C\_PLOT\_TYPE\_EP.
- **p\_window** (*int, optional*) – Moving average parameter. The default is 100.
- **p\_showing** (*Bool, optional*) – Showing graphs after they are generated. The default is True.
- **p\_printing** (*dict, optional*) – Additional important parameters for plotting. [0] = Bool : Whether the stored values is plotted. [1] = Float : Min. value on graph. [2] = Float : Max. value on graph. Set to -1, if you want to set min/max value according to the stored values. Example = {"p\_variable\_1": [True,0,-1],

"p\_variable\_2" : [True,-0.5,10]}.

The default is None.

- **p\_figsize** (*int, optional*) – Frame size. The default is (7,7).
- **p\_color** (*str, optional*) – Line colors. The default is "darkblue".
- **p\_window\_type** (*str, optional*) – Plotting type for moving average. The default is 'same'. Options: 'same', 'full', 'valid'

#### **C\_PLOT\_TYPE\_CY**

one of the plotting types, which plot the graph with multiple lines according to the number of frames.

##### **Type**

str

#### **C\_PLOT\_TYPE\_EP**

one of the plotting types, which plot the graph everything in one line regardless the number of frames.

##### **Type**

str

#### **C\_PLOT\_TYPE\_EP\_M**

one of the plotting types, which plot only the mean value of each variable for each frame.

##### **Type**

str

**C\_PLOT\_TYPE\_CY** = 'Cyclic'

**C\_PLOT\_TYPE\_EP** = 'Episodic'

**C\_PLOT\_TYPE\_EP\_M** = 'Episodic Mean'

#### **get\_plots()**

A function to plot data.

#### **plots\_type\_cy()**

A function to plot data per cycle.

#### **plots\_type\_ep()**

A function to plot data per frame by extending the cyclic plots in one plot.

#### **plots\_type\_ep\_mean()**

A function to plot data per frame according to its mean value.

#### **moving\_mean(p\_inputs, p\_window)**

This method creates a series of averages of different subsets of the full data set.

##### **Parameters**

- **p\_inputs** (*list of floats*) – input dataset.
- **p\_window** (*int*) – moving average parameter.

##### **Returns**

**outputs** – transformed data set.

##### **Return type**

list of floats

**save\_plots**(*p\_path*, *p\_format*, *p\_dpi\_mul*=1)

This method is used to save generated plots.

**Parameters**

- **p\_path** (*str*) – Path where file will be saved.
- **p\_format** (*str*) – Format of the saved file. Options: 'eps', 'jpg', 'png', 'pdf', 'svg'.
- **p\_dpi\_mul** (*int*, *optional*) – Saving plots parameter. The default is 1.

**Returns**

True, if plots where saved successfully. False otherwise..

**Return type**

bool

## BF-EXCEPTIONS - Exceptions

Ver. 1.0.2 (2021-12-12)

This module provides exception classes.

**exception** mlpro.bf.exceptions.ParamError

Bases: Exception

To be raised on a parameter error...

**exception** mlpro.bf.exceptions.ImplementationError

Bases: Exception

To be raised on an implementation error in a child class of MLPro...

**exception** mlpro.bf.exceptions.Error

Bases: Exception

To be raised on an error...

## BF-UI-SCIUI - SciUI Application Class

Ver. 0.6.0 (2021-07-05)

Provides the SciUI application class.

**class** mlpro.bf.ui.sciui.main.SciUI(*p\_fullscreen=False*, *p\_autoscan\_scenarios=True*,  
*p\_start\_immediately=True*, *p\_logging=False*)

Bases: *SciUIWindow*

**C\_TYPE** = 'Application'

**C\_NAME** = 'SciUI'

**C\_VERSION** = '0.6.0'

**\_\_init\_shared\_db**()

**\_\_init\_main\_menu**()

**\_\_init\_main\_window**()

```
start()
register_scenario(p_scenario: SciUIScenario, p_recursive=False, p_sort=True)
__register_scenario_rec(p_scenario: SciUIScenario, p_recursive)
__change_scenario()
__set_fullscreen(p_fullscreen)
__cb_global_refresh(*args)
__cb_toggle_fullscreen(*args)
__cb_menu_file_properties()
__cb_menu_help_about()
__cb_submenu_file_change_scenario()
__cb_window_resized(event)
```



```
C_TYPE = 'SciUI Window'
```

```
C_NAME = '????'
```

```
start()
```

```
class mlpro.bf.ui.sciui.framework.SciUICursor(ax, horizOn=True, vertOn=True, useblit=False,
                                              **lineprops)
```

Bases: [SciUIRoot](#), [Cursor](#)

Enriched matplotlib cursor widget.

```
connect_event(event, callback)
```

Connect a callback function with an event.

This should be used in lieu of `figure.canvas.mpl_connect` since this function stores callback ids for later clean up.

```
set_event_status(event, status)
```

```
onmove(event)
```

Internal event handler to draw the cursor when the mouse moves.

```
onbuttonpressed(event)
```

```
onbuttonreleased(event)
```

```
class mlpro.bf.ui.sciui.framework.SciUITooltip(widget, text='widget info')
```

Bases: [object](#)

Enriched tooltip class.

```
enter(event=None)
```

```
leave(event=None)
```

```
schedule()
```

```
unschedule()
```

```
showtip(event=None)
```

```
hidetip()
```

```
class mlpro.bf.ui.sciui.framework.SciUIComponent(p_shared_db: SciUISharedDB, p_logging=True)
```

Bases: [SciUIRoot](#)

Elementary screen object in SciUI framework.

```
C_TYPE = 'SciUI Component'
```

```
C_NAME = ''
```

```
init_component()
```

Initialization of component-specific elements at instance creation time. To be redefined.

```
get_name()
```



**refresh**(*p\_parent\_frame=None*)

Refresh of all component-specific elements. To be redefined. Please call `super().refresh()` at the beginning of your own implementation.

**Parameters**

**object** (*p\_parent\_frame* *Parent frame*)

```
class mlpro.bf.ui.sciui.framework.SciUIFrame(p_shared_db: SciUISharedDB, p_row, p_col,
                                             p_title=None, p_width_perc=0.0, p_height_perc=0.0,
                                             p_visible=False, p_padx=5, p_pady=0, p_sticky='NW',
                                             p_logging=True)
```

Bases: [SciUIComponent](#)

Enriched wrapper class for the Tkinter (Label-)Frame class, based on the Tkinter grid positioning model.

**C\_TYPE** = 'SciUI Frame'

**C\_NAME** = ''

**init\_popup\_menu**()

Initializes popup menu of the component. To be redefined.

**init\_component**()

Initialization of component-specific elements at instance creation time. To be redefined.

**cb\_popup\_menu**(*p\_event*)

**determine\_frame\_size**()

**refresh**(*p\_parent\_frame=None*)

Refresh of all component-specific elements. To be redefined. Please call `super().refresh()` at the beginning of your own implementation.

**Parameters**

**object** (*p\_parent\_frame* *Parent frame*)

**add\_component**(*p\_component: SciUIComponent*)

```
class mlpro.bf.ui.sciui.framework.SciUITabCTRL(p_shared_db: SciUISharedDB, p_row, p_col,
                                              p_title=None, p_width_perc=0.0, p_height_perc=0.0,
                                              p_visible=False, p_padx=5, p_pady=0,
                                              p_sticky='NW', p_logging=True)
```

Bases: [SciUIFrame](#)

Enriched wrapper class for the Tkinter tab control.

**C\_TYPE** = 'SciUI Tabs'

**init\_component**()

Initialization of component-specific elements at instance creation time. To be redefined.

**refresh**(*p\_parent\_frame*)

Refresh of all component-specific elements. To be redefined. Please call `super().refresh()` at the beginning of your own implementation.

**Parameters**

**object** (*p\_parent\_frame* *Parent frame*)

**add\_component**(*p\_component: SciUIComponent*)

```
add_tab(p_tab_name, p_component: SciUIComponent)
```

```
class mlpro.bf.ui.sciui.framework.SciUISubplotSaveDLG(p_fname=None, p_xpos=500, p_ypos=300,  
                                                       p_logging=False)
```

Bases: *SciUIWindow*

Small SciUI window application to choose folder and file name for saving a SciUI subplot.

```
C_NAME = 'Save Plot'
```

```
C_FONT_FAMILY = 'Lucida Grande'
```

```
C_FONT_SIZE = 10
```

```
C_FILENAME = 'myplot'
```

```
C_SUFFIXES = ['.pdf', '.png', '.svg']
```

```
get_filename()
```

```
__cb_button_ok()
```

```
__cb_button_cancel()
```

```
__cb_button_folder()
```

```
class mlpro.bf.ui.sciui.framework.SciUISubplotRoot(p_shared_db: SciUISharedDB, p_row, p_col,  
                                                    p_title=None, p_width_perc=0.0,  
                                                    p_height_perc=0.0, p_visible=False, p_padx=5,  
                                                    p_pady=0, p_sticky='NW', p_logging=True)
```

Bases: *SciUIFrame*

Root class for specialized frame classes that embedd a matplotlib figure with one subplot into a Tkinter frame. Not intended for direct reuse. Please use inherited classes SciUISubplot2D, SciUISubplot3D instead.

```
C_TYPE = 'SciUI Subplot'
```

```
C_BACKEND = 'TkAgg'
```

```
C_FIG_FACECOLOR = 'white'
```

```
C_AX_RECTANGLE = [0.1, 0.1, 0.85, 0.85]
```

```
C_AX_FRAME = True
```

```
C_AX_FACECOLOR = 'white'
```

```
create_subplot()
```

Internally used to create a suitable subplot. New subplot needs to be bound to self.ax. To be redefined.

```
init_popup_menu()
```

Initializes popup menu of the component. To be redefined.

```
init_component()
```

Initialization of component-specific elements at instance creation time. To be redefined. Please call super().init\_component() at beginning of your implementation.

```
determine_frame_size()
```

```
set_flush_events(p_flush: bool)
```

**refresh**(*p\_parent\_frame*)

Refresh of all component-specific elements. To be redefined. Please call `super().refresh()` at the beginning of your own implementation.

**Parameters**

**object** (*p\_parent\_frame* *Parent frame*)

**refresh\_custom**()

Additional refresh activities. To be redefined.

**add\_component**(*p\_component*)

Adding further subcomponents is disabled here.

**cb\_save\_plot**()

```
class mlpro.bf.ui.sciui.framework.SciUISubplot2D(p_shared_db: SciUISharedDB, p_row, p_col,
                                                p_title=None, p_width_perc=0.0,
                                                p_height_perc=0.0, p_visible=False, p_padx=5,
                                                p_pady=0, p_sticky='NW', p_logging=True)
```

Bases: *SciUISubplotRoot*

Specialized frame class that embeds a 2D matplotlib figure with one subplot into a Tkinter frame. A cross hair cursor functionality with mouse event handling can be switched on/off via class constant `C_AX_CURSOR`.

**C\_TYPE** = 'SciUI 2D Subplot'

**C\_AX\_RECTANGLE** = [0.1, 0.1, 0.85, 0.85]

**C\_AX\_CURSOR** = False

**C\_AX\_CURSOR\_COLOR** = 'blue'

**create\_subplot**()

Internally used to create a suitable subplot. New subplot needs to be bound to `self.ax`. To be redefined.

**\_\_init\_cursor**()

**cb\_mbutton\_pressed**(*event*)

**cb\_mouse\_moved**(*event*)

**cb\_mbutton\_released**(*event*)

```
class mlpro.bf.ui.sciui.framework.SciUISubplot3D(p_shared_db: SciUISharedDB, p_row, p_col,
                                                p_title=None, p_width_perc=0.0,
                                                p_height_perc=0.0, p_visible=False, p_padx=5,
                                                p_pady=0, p_sticky='NW', p_logging=True)
```

Bases: *SciUISubplotRoot*

Specialized frame class that embeds a 3D matplotlib figure with one subplot into a Tkinter frame.

**C\_TYPE** = 'SciUI 3D Subplot'

**C\_AX\_RECTANGLE** = [0.1, 0.1, 0.85, 0.85]

**create\_subplot**()

Internally used to create a suitable subplot. New subplot needs to be bound to `self.ax`. To be redefined.

```
class mlpro.bf.ui.sciui.framework.SciUIFrameParam(p_shared_db: SciUISharedDB, p_row, p_col,  
                                                  p_title=None, p_width_perc=0.0,  
                                                  p_height_perc=0.0, p_visible=False, p_padx=5,  
                                                  p_pady=0, p_sticky='NW', p_logging=True)
```

Bases: *SciUIFrame*

Class for parameter/text frames.

**C\_TYPE** = 'SciUI Parameter Frame'

**C\_NAME** = ''

**C\_FONT\_FAMILY** = 'Lucida Grande'

**C\_FONT\_SIZE** = 10

**init\_component()**

Initialization of component-specific elements at instance creation time. To be redefined.

```
class mlpro.bf.ui.sciui.framework.SciUIScenario(p_shared_db: SciUISharedDB, p_logging=True)
```

Bases: *SciUIFrame*

Top level class for an entire SciUI scenario that can be registered by the SciUI application class. SciUI scenarios are visible and chooseable if the switches **C\_RELEASED** and **C\_VISIBLE** are set to True.

**C\_TYPE** = 'SciUI Scenario'

**C\_NAME** = '????'

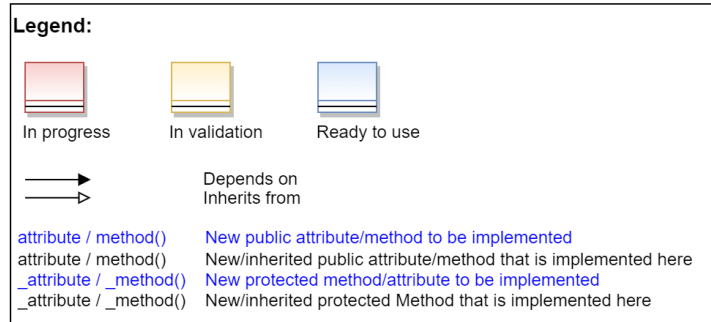
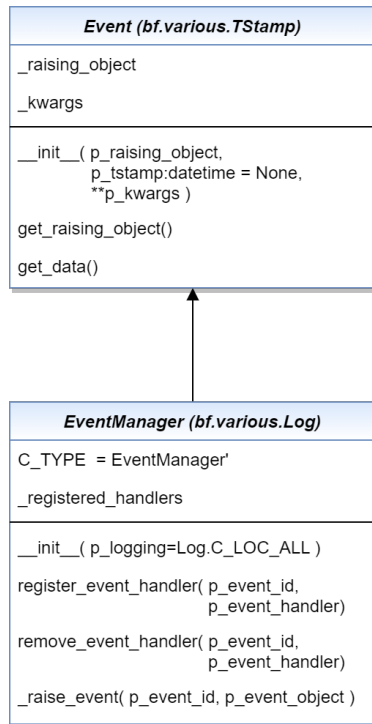
**C\_VERSION** = '0.0.0'

**C\_RELEASED** = False

**C\_VISIBLE** = False

## Layer 1 - Computation

### BF-EVENTS - Event Handling



Ver. 1.2.1 (2023-11-18)

This module provides classes for event handling. To this regard, the property class Eventmanager is provided to add event functionality to child classes by inheritance.

```
class mlpro.bf.events.Event(p_raising_object, p_tstamp: datetime = None, **p_kwargs)
```

Bases: *TStamp*

Root class for events. It is ready to use and transfers the raising object and further key/value data to the event handler.

#### Parameters

- **p\_raising\_object** – Reference to object that raised the event.
- **\*\*p\_kwargs** – List of named parameters

```
get_raising_object()
```

**get\_data()**

**class** mlpro.bf.events.**EventManager**(*p\_logging=True*)

Bases: [Log](#)

This property class provides universal event management functionalities to be inherited to child classes.

**Parameters**

**p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL

**C\_TYPE** = 'EventManager'

**register\_event\_handler**(*p\_event\_id: str, p\_event\_handler*)

Registers an event handler.

**Parameters**

- **p\_event\_id** (*str*) – Unique event id
- **p\_event\_handler** – Reference to an event handler method with parameters *p\_event\_id* and *p\_event\_object: Event*

**remove\_event\_handler**(*p\_event\_id: str, p\_event\_handler*)

Removes an already registered event handler.

**Parameters**

- **p\_event\_id** – Unique event id
- **p\_event\_handler** – Reference to an event handler method.

**\_raise\_event**(*p\_event\_id: str, p\_event\_object: [Event](#)*)

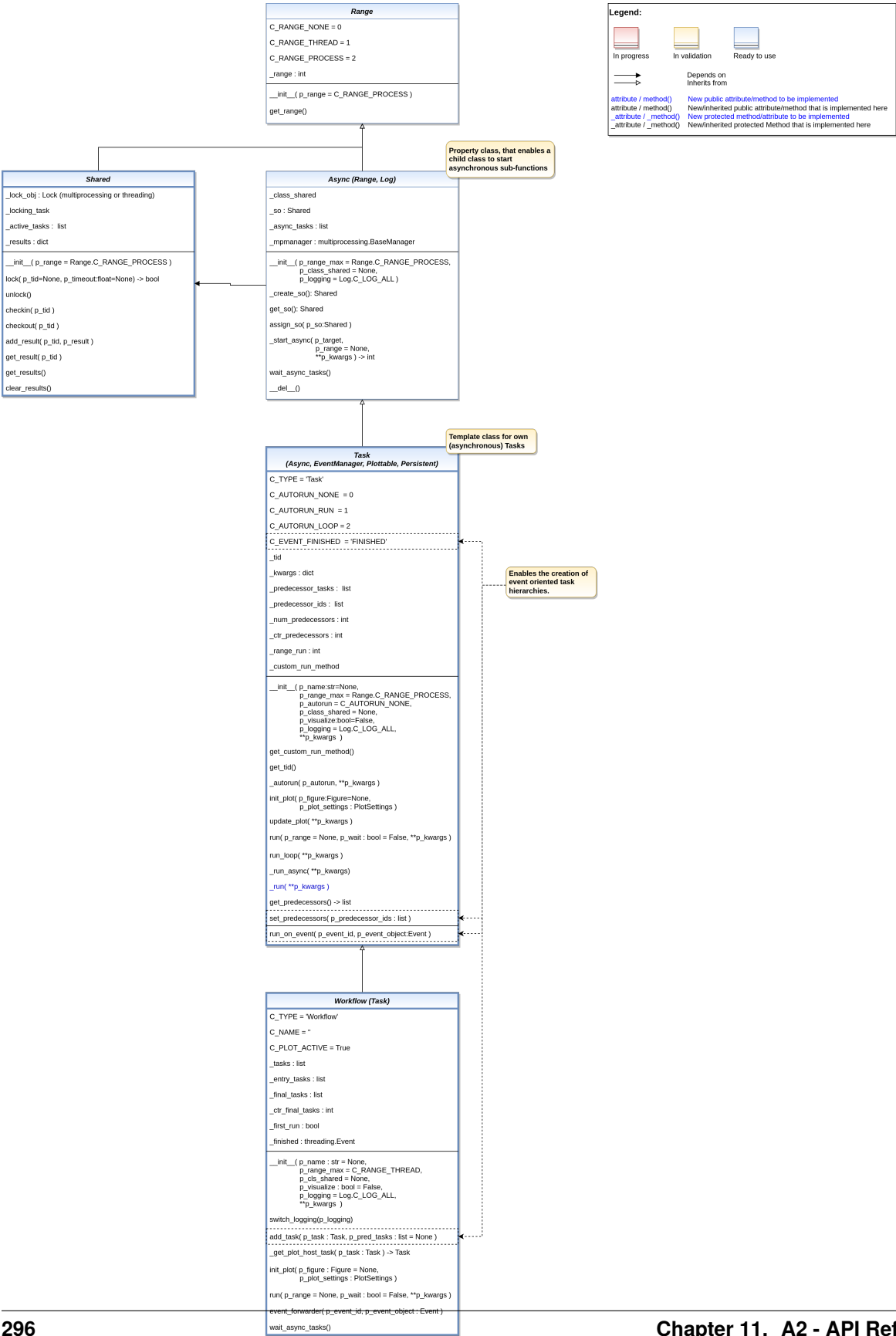
Raises an event and calls all registered handlers. To be used inside an event manager class.

**Parameters**

- **p\_event\_id** (*str*) – Unique event id
- **p\_event\_object** ([Event](#)) – Event object with further context informations



BF-MT - Multitasking





This module provides classes for multitasking with optional interprocess communication (IPC) based on shared objects. Multitasking in MLPro combines multithreading and multiprocessing and simplifies parallel programming.

Annotation to multitasking: Standard Python package multiprocessing uses pickle for serialization. This leads to problems with more complex objects. That was the reason to opt for the more flexible package multiprocess, which is a fork of multiprocessing and uses dill for serialization.

See also: <https://stackoverflow.com/questions/40234771/replace-pickle-in-python-multiprocessing-lib>

```
class mlpro.bf.mt.Range(p_range: int = 2)
```

Bases: object

Property class that defines the possible ranges of asynchronous execution supported by MLPro.

**Parameters**

**p\_range** (int) – Range of asynchronicity

**C\_RANGE\_NONE**

Synchronous execution only.

**Type**

int

**C\_RANGE\_THREAD**

Asynchronous execution as separate thread within the current process.

**Type**

int

**C\_RANGE\_PROCESS**

Asynchronous execution as separate process within the current machine.

**Type**

int

**C\_VALID\_RANGES**

List of valid ranges.

**Type**

list

**C\_RANGE\_NONE = 0**

**C\_RANGE\_THREAD = 1**

**C\_RANGE\_PROCESS = 2**

**C\_VALID\_RANGES = [0, 1, 2]**

**get\_range()** → int

```
class mlpro.bf.mt.Shared(p_range: int = 2)
```

Bases: [Range](#)

Template class for shared objects. It is ready to use and the default class for IPC. It provides elementary mechanisms for access control and messaging.

It is also possible to inherit and enrich a child class for special needs but please beware that at least in multiprocessing mode (p\_range=Range.C\_RANGE\_PROCESS) a direct access to attributes is not possible. Child classes should generally provide suitable methods for access to attributes.

**Parameters**

**p\_range** (int) – Range of asynchronicity

**C\_MSG\_TYPE\_DATA = 0**

**C\_MSG\_TYPE\_TERM = 1**

**lock**(*p\_tid=None, p\_timeout: float = None*) → bool

Locks the shared object for a specific process.

**Parameters**

- **p\_tid** – Unique task id. If None then the internal locking mechanism is disabled.
- **p\_timeout** (*float*) – Optional timeout in seconds. If None, timeout is infinite.
- **Returns**
- **True**
- **otherwise.** (*if shared object was locked successfully. False*)

**unlock**()

Unlocks the shared object.

**checkin**(*p\_tid*)

Registers a task.

**Parameters**

**p\_tid** – Task id.

**checkout**(*p\_tid*)

Unregisters a task.

**Parameters**

**p\_tid** – Task id.

**add\_result**(*p\_tid, p\_result*)

Adds a result for a task.

**Parameters**

- **p\_tid** – Task id.
- **p\_result** – Any kind of result data.

**get\_result**(*p\_tid*)

Returns the result data of a task.

**Parameters**

**p\_tid** – Task id.

**Returns**

Result data of a task.

**Return type**

task\_results

**get\_results**()

Returns reference to internal dictionary of results

**Returns**

**results** – Dictionary of results

**Return type**

dict

**clear\_results()**

Clears internal dictionary of results

**class** mlpro.bf.mt.**Async**(*p\_range\_max*: int = 2, *p\_class\_shared*=None, *p\_logging*=True)

Bases: [Range](#), [Log](#)

Property class that enables child classes to run sub-tasks asynchronously. Depending on the given range a task can be executed as a separate thread in the same process or a separate process on the same machine.

**Parameters**

- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class [Range](#). Default is `Range.C_RANGE_PROCESS`.
- **p\_class\_shared** – Optional class for a shared object (class [Shared](#) or a child class of [Shared](#))
- **p\_logging** – Log level (see constants of class [Log](#)). Default: `Log.C_LOG_ALL`

**\_create\_so**(*p\_range*: int, *p\_class\_shared*) → [Shared](#)

Internal use. Creates a suitable shared object for the given range.

**Parameters**

- **p\_range** (*int*) – Maximum range of asynchronicity. See class [Range](#). Default is `Range.C_RANGE_PROCESS`.
- **p\_class\_shared** – Class for a shared object (class [Shared](#) or a child class of [Shared](#))

**Returns**

**so** – A new shared object

**Return type**

[Shared](#)

**get\_so**() → [Shared](#)

Returns the associated shared object.

**Returns**

**so** – Shared object of type [Shared](#) (or inherited)

**Return type**

[Shared](#)

**assign\_so**(*p\_so*: [Shared](#))

Assigns an existing shared object to the task. The task takes over the range of asynchronicity of the shared object if it is less than the current one of the task.

**Parameters**

**p\_so** ([Shared](#)) – Shared object.

**\_start\_async**(*p\_target*, *p\_range*: int = None, *\*\*p\_kwargs*) → int

Starts a method or a new instance of a given class asynchronously. If neither a method nor a class is specified, a new instance of the current class is created asynchronously.

**Parameters**

- **p\_target** – A class, method or function to be executed (a)synchronously depending on the actual range
- **p\_range** (*int*) – Optional deviating range of asynchronicity. See class [Range](#). Default is None what means that the maximum range defined during instantiation is taken. Otherwise the minimum range of both is taken.

- **p\_kwargs** (*dictionary*) – Parameters to be handed over to asynchronous method/instance

**Returns**

**range** – Actual range of asynchronicity

**Return type**

int

**wait\_async\_tasks()**

Waits until all internal asynchronous tasks are finished.

```
class mlpro.bf.mt.Task(p_id=None, p_name: str = None, p_range_max: int = 1, p_autorun=0,
                      p_class_shared=None, p_visualize: bool = False, p_logging=True, **p_kwargs)
```

Bases: [Async](#), [EventManager](#), [Plottable](#), [Persistent](#)

Template class for a task, that can run things - and even itself - asynchronously in a thread or process. Tasks can run standalone or as part of a workflow (see class Workflow). The integrated event manager allows callbacks on specific events inside the same process(!).

**Parameters**

- **p\_id** – Optional external id
- **p\_name** (*str*) – Optional name of the task. Default is None.
- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class Range. Default is Range.C\_RANGE\_THREAD.
- **p\_autorun** (*int*) – On value C\_AUTORUN\_RUN method run() is called immediately during instantiation. On value C\_AUTORUN\_LOOP method run\_loop() is called. Value C\_AUTORUN\_NONE (default) causes an object instantiation without starting further actions.
- **p\_class\_shared** – Optional class for a shared object (class Shared or a child class of Shared)
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL
- **p\_kwargs** (*dict*) – Further optional named parameters.

C\_TYPE = 'Task'

C\_AUTORUN\_NONE = 0

C\_AUTORUN\_RUN = 1

C\_AUTORUN\_LOOP = 2

C\_EVENT\_FINISHED = 'FINISHED'

**\_get\_custom\_run\_method()**

**get\_tid()**

Returns unique task id.

**\_autorun**(*p\_autorun*, *\*\*p\_kwargs*)

Internal method to automate a single or looped run.

**Parameters**

- **p\_autorun** (*int*) – On value C\_AUTORUN\_RUN method run() is called immediately during instantiation. On value C\_AUTORUN\_LOOP method run\_loop() is called. Value C\_AUTORUN\_NONE (default) causes an object instantiation without starting further actions.
- **p\_kwargs** (*dict*) – Further parameters handed over to method run().

**run**(*p\_range: int = None, p\_wait: bool = False, \*\*p\_kwargs*)

Executes the task specific actions implemented in custom method \_run(). At the end event C\_EVENT\_FINISHED is raised to start subsequent actions (p\_wait=True).

#### Parameters

- **p\_range** (*int*) – Optional deviating range of asynchronicity. See class Range. Default is None what means that the maximum range defined during instantiation is taken. Otherwise the minimum range of both is taken.
- **p\_wait** (*bool*) – If True, the method waits until all (a)synchronous tasks are finished.
- **p\_kwargs** (*dict*) – Further parameters handed over to custom method \_run().

**\_run\_async**(*\*\*p\_kwargs*)

Internally used by method run(). It runs the custom method \_run() and raises event C\_EVENT\_FINISHED.

#### Parameters

- **p\_kwargs** (*dict*) – Custom parameters.

**\_run**(*\*\*p\_kwargs*)

Custom method that is called (asynchronously) by method run().

#### Parameters

- **p\_kwargs** (*dict*) – Custom parameters.

**run\_loop**(*\*\*p\_kwargs*)

Executes method run() in a loop, until a message of type Shared.C\_MSG\_TYPE\_TERM is sent to the task.

#### Parameters

- **p\_kwargs** (*dict*) – Parameters for method run()

**get\_predecessors**() → list

**set\_predecessors**(*p\_predecessor\_tasks: list*)

Used by class Workflow to inform a task about its number of predecessor tasks. See method run\_on\_event().

#### Parameters

- **p\_predecessor\_ids** (*list*) – List of ids of predecessor tasks in a workflow.

**run\_on\_event**(*p\_event\_id, p\_event\_object: Event*)

Can be used as event handler - in particular for other tasks in a workflow in combination with event C\_EVENT\_FINISHED. Method self.run() is called if the last predecessor task in a workflow has raised event C\_EVENT\_FINISHED.

#### Parameters

- **p\_event\_id** – Event id.
- **p\_event\_object** (*Event*) – Event object with further context informations.

**init\_plot**(*p\_figure: Figure = None, p\_plot\_settings: PlotSettings = None*)

Initializes the plot functionalities of the class.

#### Parameters

- **p\_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p\_plot\_settings** (*PlotSettings*) – Optional plot settings. If None, the default view is plotted (see attribute C\_PLOT\_DEFAULT\_VIEW).

**update\_plot**(\*\*p\_kwargs)

Updates the plot.

**Parameters**

**\*\*p\_kwargs** – Implementation-specific plot data and/or parameters.

**class** mlpro.bf.mt.**Workflow**(p\_name: str = None, p\_range\_max=1, p\_class\_shared=None, p\_visualize: bool = False, p\_logging=True, \*\*p\_kwargs)

Bases: *Task*

Ready-to-use container class for task groups. Objects of type Task (or inherited) can be added and chained to sequences or hierarchies of tasks.

**Parameters**

- **p\_name** (str) – Optional name of the task. Default is None.
- **p\_range\_max** (int) – Range of asynchronicity. See class Range. Default is Range.C\_RANGE\_THREAD.
- **p\_class\_shared** – Optional class for a shared object (class Shared or a child class of Shared)
- **p\_visualize** (bool) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL
- **p\_kwargs** (dict) – Further optional named parameters handed over to every task within.

C\_TYPE = 'Workflow'

C\_NAME = ''

C\_PLOT\_ACTIVE: bool = True

**switch\_logging**(p\_logging)

Sets log level for the workflow and all tasks inside.

**Parameters**

**p\_logging** – Log level (see constants of class Log).

**add\_task**(p\_task: Task, p\_pred\_tasks: list = None)

Adds a task to the workflow.

**Parameters**

- **p\_task** (Task) – Task object to be added.
- **p\_pred\_tasks** (list) – Optional list of predecessor task objects

**\_get\_plot\_host\_task**(p\_task: Task) → Task

**init\_plot**(p\_figure: Figure = None, p\_plot\_settings: PlotSettings = None)

Initializes the plot of a workflow. The method creates a host figure for all tasks if no external host figure is parameterized. The sub-plots of the tasks are automatically arranged within the host figure.

See method init\_plot() of class mlpro.bf.plot.Plottable for further details.

### Parameters

- **p\_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p\_plot\_settings** (*PlotSettings*) – Optional plot settings. If None, the default view is plotted (see attribute C\_PLOT\_DEFAULT\_VIEW).

**run**(*p\_range: int = None, p\_wait: bool = False, \*\*p\_kwargs*)

Executes all tasks of the workflow. At the end event C\_EVENT\_FINISHED is raised to start subsequent actions (*p\_wait=True*).

### Parameters

- **p\_range** (*int*) – Optional deviating range of asynchronicity. See class Range. Default is None what means that the maximum range defined during instantiation is taken. Otherwise the minimum range of both is taken.
- **p\_wait** (*bool*) – If True, the method waits until all (a)synchronous tasks are finished.
- **p\_kwargs** (*dict*) – Further parameters handed over to custom method `_run()`.

**event\_forwarder**(*p\_event\_id, p\_event\_object: Event*)

Internally used to raise event C\_EVENT\_FINISHED on workflow level if all final tasks have been finished.

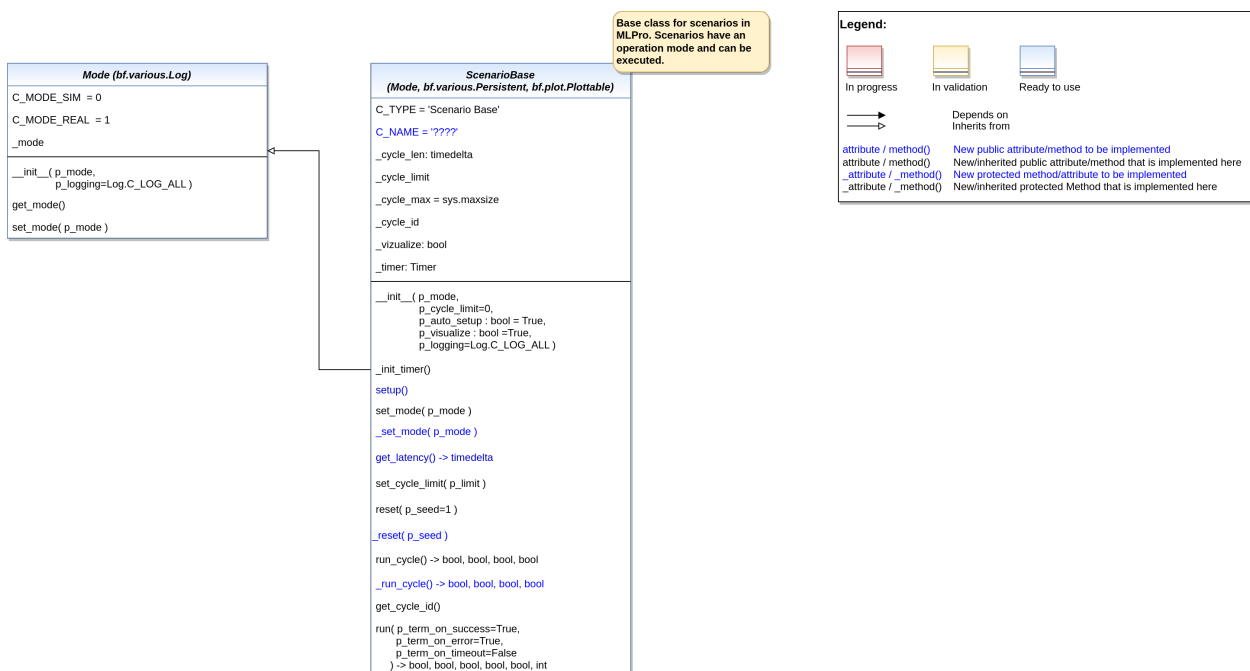
### Parameters

- **p\_event\_id** – Event id.
- **p\_event\_object** (*Event*) – Event object with further context informations.

**wait\_async\_tasks()**

Waits until all tasks are finished.

## BF-OPS - Operations



Ver. 1.2.3 (2023-03-25)

This module provides classes for operation.

**class** mlpro.bf.ops.**Mode**(*p\_mode*, *p\_logging*=True)

Bases: [Log](#)

Property class that adds a mode and related methods to a child class.

#### Parameters

- **p\_mode** – Operation mode. Valid values are stored in constant C\_VALID\_MODES.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL

**C\_MODE\_SIM** = 0

Simulation mode.

**C\_MODE\_REAL** = 1

Real operation mode.

**C\_VALID\_MODES**

List of valid modes.

#### Type

list

**C\_MODE\_INITIAL** = -1

**C\_MODE\_SIM** = 0

**C\_MODE\_REAL** = 1

**C\_VALID\_MODES** = [0, 1]

**get\_mode()**

Returns current mode.

**set\_mode**(*p\_mode*)

Sets new mode.

#### Parameters

- **p\_mode** – Operation mode. Valid values are stored in constant C\_VALID\_MODES.

**class** mlpro.bf.ops.**ScenarioBase**(*p\_mode*, *p\_id*=None, *p\_cycle\_limit*=0, *p\_auto\_setup*: bool = True, *p\_visualize*: bool = True, *p\_logging*=True)

Bases: [Mode](#), [Persistent](#), [Plottable](#)

Base class for executable scenarios in MLPro. To be inherited and specialized in higher layers.

**The following key features are included:**

- Operation mode
- Cycle management
- Timer
- Latency

#### Parameters

- **p\_mode** – Operation mode. See Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM.
- **p\_id** – Optional external id



- **p\_cycle\_limit** (*int*) – Maximum number of cycles. Default = 0 (no limit).
- **p\_auto\_setup** (*bool*) – If True custom method setup() is called after initialization.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = True.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL.

**C\_TYPE** = 'Scenario Base'

**C\_NAME** = '????'

**\_init\_timer()**

**setup()**

Custom method to set up all components of the scenario.

**set\_mode**(*p\_mode*)

Sets operation mode of the scenario. Custom method \_set\_mode() is called.

### Parameter

**p\_mode**

Operation mode. See class bf.ops.Mode for further details.

**\_set\_mode**(*p\_mode*)

Custom method to set the operation mode of components of the scenario. See method set\_mode() for further details.

### Parameter

**p\_mode**

Operation mode. See class bf.ops.Mode for further details.

**get\_latency()** → *timedelta*

Returns the latency of the scenario. To be implemented in child class.

**set\_cycle\_limit**(*p\_limit*)

Sets the maximum number of cycles to run.

### Parameters

**p\_cycle\_limit** (*int*) – Maximum number of cycles. Default = 0 (no limit).

**reset**(*p\_seed=1*)

Resets the scenario and especially the ML model inside. Internal random generators are seed with the given value. Custom reset actions can be implemented in method \_reset().

### Parameters

**p\_seed** (*int*) – Seed value for internal random generator

**\_reset**(*p\_seed*)

Custom method to reset the components of the scenario and to set the given random seed value. See method reset() for further details.

### Parameters

**p\_seed** (*int*) – Seed value for internal random generator

**run\_cycle()**

Runs a single process cycle.

**Returns**

- **success** (*bool*) – True on success. False otherwise.
- **error** (*bool*) – True on error. False otherwise.
- **timeout** (*bool*) – True on timeout. False otherwise.
- **cycle\_limit** (*bool*) – True, if cycle limit has reached. False otherwise.
- **adapted** (*bool*) – True, if something within the scenario has adapted something in this cycle. False otherwise.
- **end\_of\_data** (*bool*) – True, if the end of the related data source has been reached. False otherwise.

**\_run\_cycle()**

Custom implementation of a single process cycle. To be redefined.

**Returns**

- **success** (*bool*) – True on success. False otherwise.
- **error** (*bool*) – True on error. False otherwise.
- **adapted** (*bool*) – True, if something within the scenario has adapted something in this cycle. False otherwise.
- **end\_of\_data** (*bool*) – True, if the end of the related data source has been reached. False otherwise.

**get\_cycle\_id()**

Returns current cycle id.

**run**(*p\_term\_on\_success: bool = True, p\_term\_on\_error: bool = True, p\_term\_on\_timeout: bool = False*)

Runs the scenario as a sequence of single process steps until a terminating event occurs.

**Parameters**

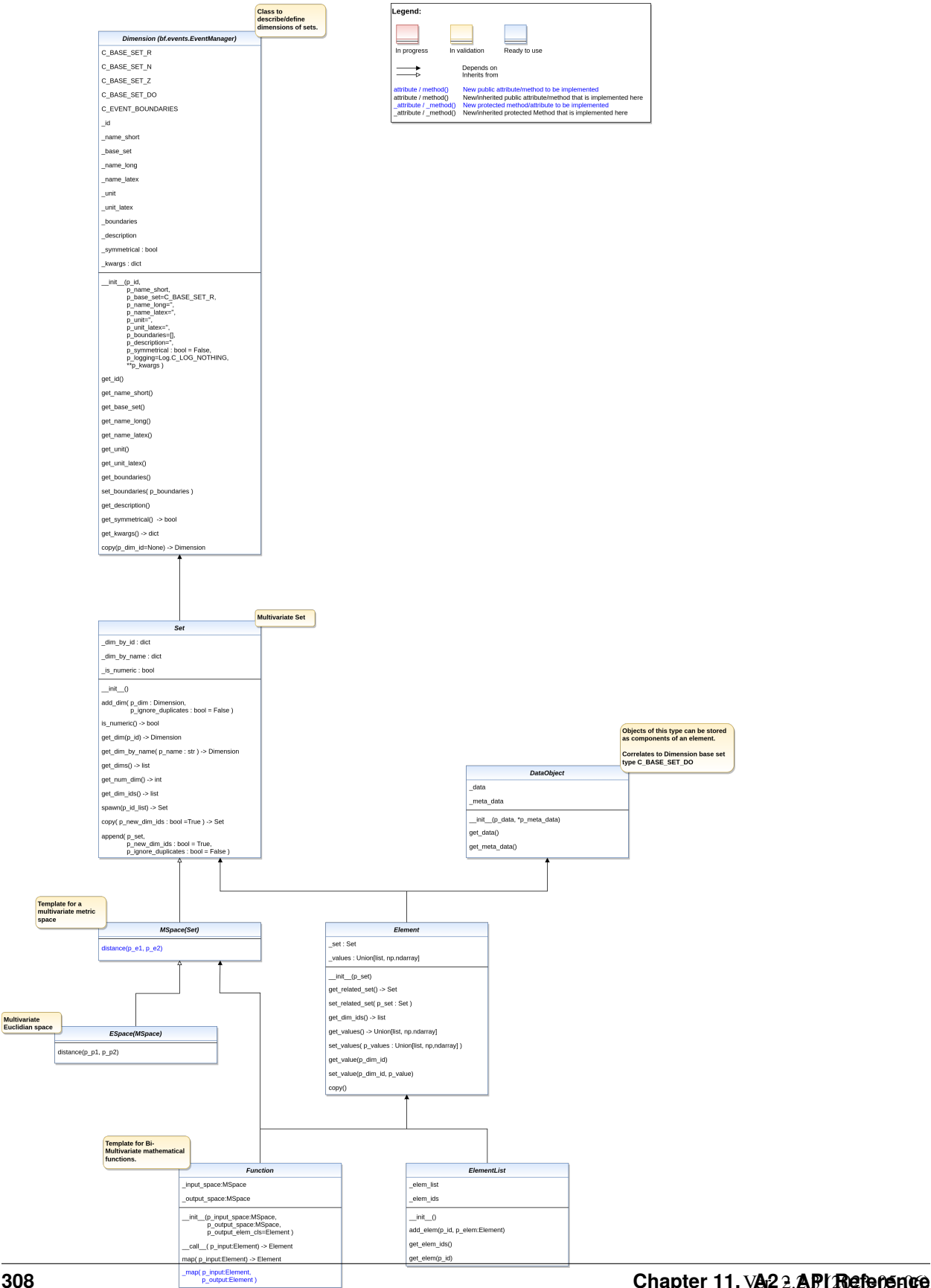
- **p\_term\_on\_success** (*bool*) – If True, the run terminates on success. Default = True.
- **p\_term\_on\_error** (*bool*) – If True, the run terminates on error. Default = True.
- **p\_term\_on\_timeout** (*bool*) – If True, the run terminates on timeout. Default = False.

**Returns**

- **success** (*bool*) – True on success. False otherwise.
- **error** (*bool*) – True on error. False otherwise.
- **timeout** (*bool*) – True on timeout. False otherwise.
- **cycle\_limit** (*bool*) – True, if cycle limit has reached. False otherwise.
- **adapted** (*bool*) – True, if ml model adapted something. False otherwise.
- **end\_of\_data** (*bool*) – True, if the end of the related data source has been reached. False otherwise.
- **num\_cycles** (*int*) – Number of cycles.

## Layer 2 - Mathematics

BF-MATH - Basic Mathematics



This module provides basic mathematical classes.

```
class mlpro.bf.math.basics.Dimension(p_name_short, p_base_set='R', p_name_long='', p_name_latex='',
                                     p_unit='', p_unit_latex='', p_boundaries: list = [], p_description='',
                                     p_symmetrical: bool = False, p_logging=False, **p_kwargs)
```

Bases: [EventManager](#)

Objects of this type specify properties of a dimension of a set.

### Parameters:

#### **p\_name\_short**

[str] Short name of dimension

#### **p\_base\_set**

Base set of dimension. See constants C\_BASE\_SET\_\*. Default = C\_BASE\_SET\_R.

#### **p\_name\_long :str**

Long name of dimension (optional)

#### **p\_name\_latex**

[str] LaTeX name of dimension (optional)

#### **p\_unit**

[str] Unit (optional)

#### **p\_unit\_latex**

[str] LaTeX code of unit (optional)

#### **p\_boundaries**

[list] List with minimum and maximum value (optional)

#### **p\_description**

[str] Description of dimension (optional)

#### **p\_symmetrical**

[bool] Information about the symmetry of the dimension (optional, default is False)

#### **p\_logging**

Log level (see constants of class Log). Default: Log.C\_LOG\_ALL

#### **p\_kwargs**

[dict] Further optional keyword parameters.

**C\_TYPE = 'Dimension'**

**C\_BASE\_SET\_R = 'R'**

**C\_BASE\_SET\_N = 'N'**

**C\_BASE\_SET\_Z = 'Z'**

**C\_BASE\_SET\_DO = 'DO'**

**C\_EVENT\_BOUNDARIES = 'BOUNDARIES'**

**get\_id()**

**get\_name\_short()**

`get_base_set()`

`get_name_long()`

`get_name_latex()`

`get_unit()`

`get_unit_latex()`

`get_boundaries()`

`set_boundaries(p_boundaries: list)`

Sets new boundaries with respect to the symmetry and raises event C\_EVENT\_BOUNDARIES.

**Parameters**

**p\_boundaries** (*list*) – New boundaries (lower and upper value)

`get_description()`

`get_symmetrical()` → bool

`get_kwargs()` → dict

`copy()`

**class** mlpro.bf.math.basics.Set

Bases: object

Objects of this type describe a (multivariate) set in a mathematical sense.

**C\_NUMERIC\_BASE\_SETS** = ['N', 'Z', 'R']

`add_dim(p_dim: Dimension, p_ignore_duplicates: bool = False)`

Raises the dimensionality of the set by adding a new dimension.

**Parameters**

- **p\_dim** ([Dimension](#)) – Dimension to be added.
- **p\_ignore\_duplicates** (*bool*) – If True, duplicated short names of dimensions are accepted. Default = False.

`is_numeric()` → bool

Returns True if the set consists of numeric dimensions only.

`get_dim(p_id) → Dimension`

Returns the dimension specified by it's unique id.

`get_dim_by_name(p_name) → Dimension`

`get_dims()` → list

” Returns all dimensions.

`get_num_dim()`

Returns the dimensionality of the set (=number of dimensions of the set).

`get_dim_ids()`

Returns the unique ids of the related dimensions.

**spawn**(*p\_id\_list*: list)

Spawns a new class with same type and a subset of dimensions specified by an index list.

**Parameters**

**adopted** (*p\_id\_list* List of indices of dimensions to be)

**Returns**

New object with subset of dimensions

**copy**(*p\_new\_dim\_ids*: bool = True)

**append**(*p\_set*, *p\_new\_dim\_ids*: bool = True, *p\_ignore\_duplicates*: bool = False)

**class** mlpro.bf.math.basics.**DataObject**(*p\_data*, \**p\_meta\_data*)

Bases: object

Container class for (big) data objects of any type with optional additional meta data.

**get\_data**()

**get\_meta\_data**() → tuple

**class** mlpro.bf.math.basics.**Element**(*p\_set*: Set)

Bases: object

Element of a (multivariate) set.

**Parameters**

**p\_set** (Set) – Underlying set.

**get\_related\_set**() → Set

**set\_related\_set**(*p\_set*: Set)

**get\_dim\_ids**() → list

**get\_values**() → list | ndarray

**set\_values**(*p\_values*: list | ndarray)

Overwrites the values of all components of the element.

**Parameters**

**dimensions.** (*p\_values* Something iterable with same length as number of element)

**get\_value**(*p\_dim\_id*)

**set\_value**(*p\_dim\_id*, *p\_value*)

**copy**()

**class** mlpro.bf.math.basics.**ElementList**

Bases: object

List of Element objects.

**add\_elem**(*p\_id*, *p\_elem*: Element)

Adds an element object under it's id in the internal element list.

**Parameters**

- **element** (*p\_id* Unique id of the)

- **added** (*p\_elem* *Element* object to be)

**get\_elem\_ids**() → list

**get\_elem**(*p\_id*) → *Element*

**class** mlpro.bf.math.basics.**BatchElement**(*p\_set*: Set)

Bases: *Element*

**class** mlpro.bf.math.basics.**MSpace**

Bases: Set

Objects of this type represent a metric space. The method distance implements the metric of the space.

**distance**(*p\_e1*: *Element*, *p\_e2*: *Element*)

**class** mlpro.bf.math.basics.**ESpace**

Bases: *MSpace*

Objects of this type represent an Euclidian space. The distance method implements the Euclidian norm.

**distance**(*p\_e1*: *Element*, *p\_e2*: *Element*)

**class** mlpro.bf.math.basics.**Function**(*p\_input\_space*: ~mlpro.bf.math.basics.*MSpace*, *p\_output\_space*: ~mlpro.bf.math.basics.*MSpace*, *p\_output\_elem\_cls*=<class 'mlpro.bf.math.basics.*Element*'>)

Bases: object

Model class for an elementary bi-multivariate mathematical function that maps elements of a multivariate input space to elements of a multivariate output space.

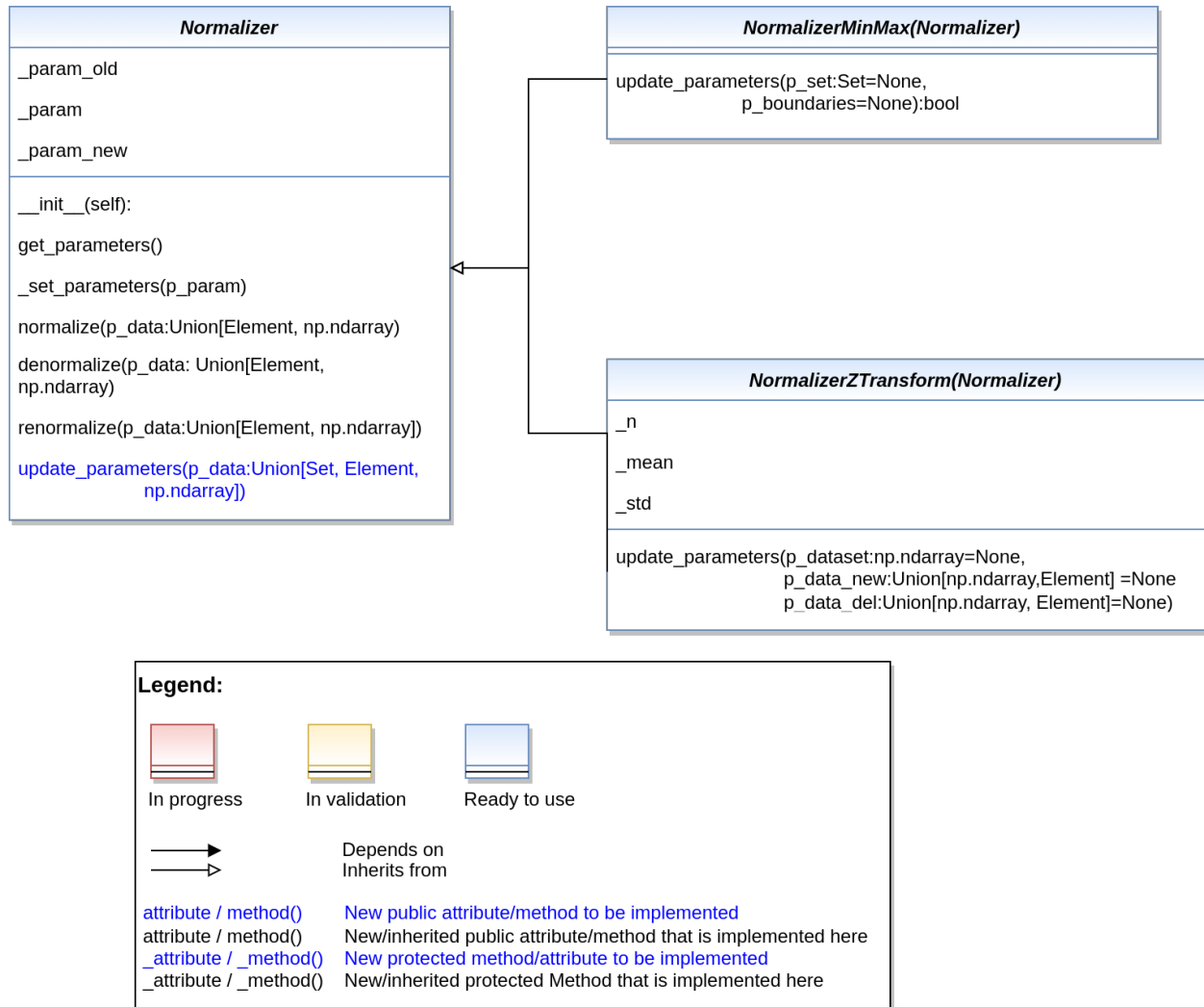
**map**(*p\_input*: *Element*) → *Element*

Alternative method to map an input to an output. Actually, it refers to method `__call__()`. Redefining this method has no effect. See method `__call__()` and constructor for further details.

**map**(*p\_input*: *Element*, *p\_output*: *Element*)

Custom method for own mapping algorithm. See methods `__call__()` and `map()` for further details.



**BF-MATH-NORMALIZERS - Normalizers**

Ver. 1.0.14 (2023-02-13) This module provides base class for Normalizers and normalizer objects including MinMax normalization and normalization by Z transformation.

**class** `mlpro.bf.math.normalizers.Normalizer`

Bases: `object`

Base template class for normalizer objects.

**`_set_parameters(p_param)`**

custom method to set the normalization parameters

**Parameters**

**`p_set (Set)`** – Set related to the elements to be normalized

**Returns**

**`boolean`** – Returns true after setting the parameters

**Return type**

True

**normalize**(*p\_data*: [Element](#) | *ndarray*)

Method to normalize a data (Element/ndarray) element based on MinMax or Z-transformation

**Parameters****p\_data** ([Element](#) or a *numpy array*) – Data element to be normalized**Returns****element** – Normalized Data**Return type**[Element](#) or *numpy array***denormalize**(*p\_data*: [Element](#) | *ndarray*)

Method to denormalize a data (Element/ndarray) element based on MinMax or Z-transformation

**Parameters****p\_data** ([Element](#) or a *numpy array*) – Data element to be denormalized**Returns****element** – Denormalized Data**Return type**[Element](#) or *numpy array***renormalize**(*p\_data*: [Element](#) | *ndarray*)

Method to denormalize and renormalize an element based on old and current normalization parameters.

**Parameters****p\_data** ([Element](#) or *numpy array*) – Element to be renormalized.**Returns****renormalized\_element** – Renormalized Data**Return type**[Element](#) or *numpy array***update\_parameters**(*p\_data*: [Set](#) | [Element](#) | *ndarray*)

Custom method to update normalization parameters.

**Parameters****p\_data** – arguments specific to normalization parameters. Check the normalizer objects for specific parameters**class** `mlpro.bf.math.normalizers.NormalizerMinMax`Bases: [Normalizer](#)

Class to normalize elements based on MinMax normalization.

**update\_parameters**(*p\_set*: [Set](#) = *None*, *p\_boundaries*: *list* | *ndarray* = *None*)

Method to update the normalization parameters of MinMax normalizer.

**Parameters**

- **p\_set** ([Set](#)) – Set related to the elements to be normalized
- **p\_boundaries** (*ndarray*) – array consisting of boundaries related to the dimension of the array

`class mlpro.bf.math.normalizers.NormalizerZTrans`

Bases: *Normalizer*

Class for Normalization based on Z transformation.

`update_parameters(p_dataset: ndarray = None, p_data_new: Element | ndarray = None, p_data_del: Element | ndarray = None)`


Method to update the normalization parameters for Z transformer


Parameters


- **p\_dataset** (*numpy array*) – Dataset related to the elements to be normalized. Using this parameter will reset the normalization parameters based on the dataset provided.
- **p\_data\_new** (*Element or numpy array*) – New element to update the normalization parameters. Using this parameter will set/update the normalization parameters based on the data provided.
- **p\_data\_del** (*Element or Numpy array*) – Old element that is replaced with the new element.


BF-MATH-GEOMETRY - Geometric Figures


Legend:

  
In progress

  
In validation

  
Ready to use





Depends on  
Inherits from

`attribute / method()`

`attribute / method()`

`attribute / _method()`

`attribute / _method()`

New public attribute/method to be implemented

New/inherited public attribute/method that is implemented here

New protected method/attribute to be implemented

New/inherited protected attribute/method that is implemented here

Point (bf.plot.Plottable)
<code>_point_pos : np.ndarray</code>
<code>_point_vel : np.ndarray</code>
<code>_point_acc : np.ndarray</code>
<code>_last_update : datetime</code>
<code>__init__( p_pos : Element = None, p_visualize : bool = False )</code>
<code>get_details() -&gt; Tuple[ np.array, np.array, np.array ]</code>
<code>set_pos( p_pos : Union[ list, np.array ], p_time_stamp : datetime = None )</code>
<code>_update_plot_2d( p_settings:PlotSettings, **p_kwargs )</code>
<code><u>_update_plot_3d( p_settings:PlotSettings, **p_kwargs )</u></code>

HyperCuboid (bf.plot.Plottable)
<code>_boundaries</code>
<code>__init__( p_pos : Element = None, p_visualize : bool = False )</code>
<code>get_point() -&gt; Tuple[ Element, Element, Element ]</code>
<code>set_point( p_pos : Element )</code>
<code>_init_plot_2d( p_figure:Figure, p_settings: PlotSettings )</code>
<code>_init_plot_3d( p_figure:Figure, p_settings: PlotSettings )</code>
<code>_init_plot_nd( p_figure:Figure, p_settings: PlotSettings )</code>
<code>_update_plot_2d( p_settings:PlotSettings, **p_kwargs )</code>
<code>_update_plot_3d( p_settings:PlotSettings, **p_kwargs )</code>
<code>_update_plot_nd( p_settings:PlotSettings, **p_kwargs )</code>

This module provides class for geometric objects like points, etc.

**class** mlpro.bf.math.geometry.**Point**(*p\_visualize: bool = False*)

Bases: Properties, [Plottable](#)

Implementation of a point in a hyper space. Properties like the current position, velocity and acceleration are managed.

**Parameters**

**p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.

**C\_PROPERTY\_POS** = 'Position'

**C\_PLOT\_ACTIVE**: **bool** = **True**

**get\_velocity()**

Returns current velocity of the point.

**Returns**

Current velocity of the point in unit/sec for each dimension.

**Return type**

point\_vel

**get\_acceleration()**

Returns current acceleration of the point.

**Returns**

Current acceleration of the point in unit/sec<sup>2</sup> for each dimension.

**Return type**

point\_acc

**get\_position()**

Returns current position of the point.

**Returns**

Current position of the point.

**Return type**

point\_pos

**set\_position**(*p\_pos: list | ndarray, p\_time\_stamp: datetime = None*)

Set/updates the point position and computes the resulting velocity and acceleration. It also updates the visualization.

**Parameters**

- **p\_pos** (*Union[ list, np.ndarray ]*) – New position of the point.
- **p\_time\_stamp** (*datetime = None*) – Optional time stamp.

**\_update\_plot\_2d**(*p\_settings: PlotSettings, \*\*p\_kwargs*)

Custom method to update the 2d plot. The related Matplotlib Axes object is stored in p\_settings.

**Parameters**

- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.
- **\*\*p\_kwargs** – Implementation-specific data and parameters.

**`_update_plot_3d`**(*p\_settings*: [PlotSettings](#), *\*\*p\_kwargs*)

Custom method to update the 3d plot. The related Matplotlib Axes object is stored in *p\_settings*.

**Parameters**

- **`p_settings`** ([PlotSettings](#)) – Object with further plot settings.
- **`**p_kwargs`** – Implementation-specific data and parameters.

**`_update_plot_nd`**(*p\_settings*: [PlotSettings](#), *\*\*p\_kwargs*)

Custom method to update the nd plot. The related Matplotlib Axes object is stored in *p\_settings*.

**Parameters**

- **`p_settings`** ([PlotSettings](#)) – Object with further plot settings.
- **`**p_kwargs`** – Implementation-specific data and parameters.

**`_remove_plot_2d`**()

Custom method to remove 2D plot artifacts when object is destroyed.

**`_remove_plot_3d`**()

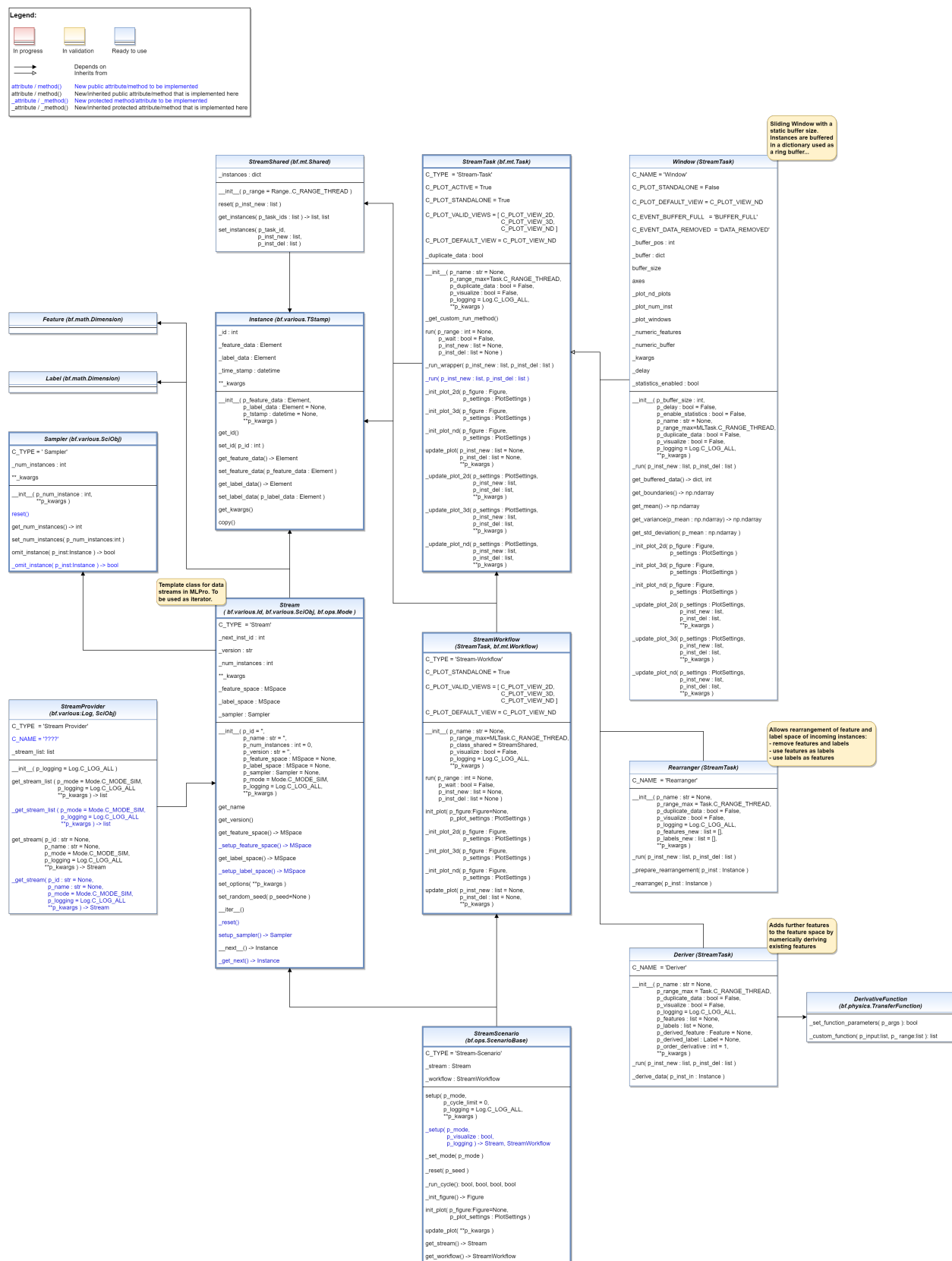
Custom method to remove 3D plot artifacts when object is destroyed.

**`_remove_plot_nd`**()

Custom method to remove nd plot artifacts when object is destroyed.

### Layer 3 - Application Support

## BF-STREAMS - Stream Processing



This module provides classes for standardized stream processing.

```
class mlpro.bf.streams.models.Feature(p_name_short, p_base_set='R', p_name_long=",
                                     p_name_latex=", p_unit=", p_unit_latex=", p_boundaries: list =
                                     [], p_description=", p_symmetrical: bool = False,
                                     p_logging=False, **p_kwargs)
```

Bases: *Dimension*

```
class mlpro.bf.streams.models.Label(p_name_short, p_base_set='R', p_name_long=", p_name_latex=",
                                     p_unit=", p_unit_latex=", p_boundaries: list = [], p_description=",
                                     p_symmetrical: bool = False, p_logging=False, **p_kwargs)
```

Bases: *Dimension*

```
class mlpro.bf.streams.models.Instance(p_feature_data: Element, p_label_data: Element = None,
                                     p_tstamp: datetime = None, **p_kwargs)
```

Bases: *TStamp*

Instance class to store the current instance and the corresponding labels of the stream

#### Parameters

- **p\_feature\_data** (*Element*) – Feature data of the instance.
- **p\_label\_data** (*Element*) – Optional label data of the instance.
- **p\_tstamp** (*datetime*) – Optional time stamp of the instance.
- **p\_kwargs** (*dict*) – Further optional named parameters.

**C\_TYPE** = 'Instance'

**get\_id()**

**set\_id**(p\_id: *int*)

**get\_feature\_data()** → *Element*

**set\_feature\_data**(p\_feature\_data: *Element*)

**get\_label\_data()** → *Element*

**set\_label\_data**(p\_label\_data: *Element*)

**get\_kwargs()**

**copy()**

```
class mlpro.bf.streams.models.StreamShared(p_range: int = 2)
```

Bases: *Shared*

Template class for shared objects in the context of stream processing.

#### **\_instances**

Dictionary of new/deleted instances per task. At the beginning of a cycle it contains the incoming instance of a stream. The dictionary evolves due to the manipulations of the stream tasks.

#### **Type**

dict

**reset**(*p\_inst\_new*: List[Instance])

Resets the shared object and prepares the processing of the given set of new instances.

**Parameters**

**p\_inst\_new** (List[Instance]) – List of new instances to be processed.

**get\_instances**(*p\_task\_ids*: list)

Provides the result instances of all given task ids.

**Parameters**

**p\_task\_ids** (list) – List of task ids.

**Returns**

- **inst\_new** (list) – List of new instances of all given task ids.
- **inst\_del** (list) – List of instances to be deleted of all given task ids.

**set\_instances**(*p\_task\_id*, *p\_inst\_new*: List[Instance], *p\_inst\_del*: List[Instance])

Stores result instances of a task in the shared object.

**Parameters**

- **p\_task\_id** – Id of related task.
- **p\_inst\_new** (list) – List of new instances.
- **p\_inst\_del** (list) – List of instances to be deleted.

**class** mlpro.bf.streams.models.Sampler(*p\_num\_instances*: int = 0, *\*\*p\_kwargs*)

Bases: [ScientificObject](#)

Template class for data streams sampler. This object can be used in Stream.

**Parameters**

- **p\_num\_instances** (int) – number of instances.
- **p\_kwargs** (dict) – Further sampler specific parameters.

**C\_TYPE** = 'Sampler'

**reset**()

A method to reset the sampler's settings. Please redefine this method!

**get\_num\_instances**() → int

A method to get the number of instances that is being processed by the sampler.

**Returns**

Number of instances.

**Return type**

int

**set\_num\_instances**(*p\_num\_instances*: int)

A method to set the number of instances that is going to be processed by the sampler.

**Parameters**

**p\_num\_instances** (int) – Number of instances.

**omit\_instance**(*p\_inst*: Instance) → bool

A method to filter any incoming instances.



**Parameters**

**p\_inst** ([Instance](#)) – An input instance to be filtered.

**Returns**

False means the input instance is not omitted, otherwise True.

**Return type**

bool

**\_omit\_instance**(*p\_inst*: [Instance](#)) → bool

A custom method to filter any incoming instances, which is being called by omit\_instance() method. Please redefine this method!

**Parameters**

**p\_inst** ([Instance](#)) – An input instance to be filtered.

**Returns**

False means the input instance is not omitted, otherwise True.

**Return type**

bool

```
class mlpro.bf.streams.models.Stream(p_id=None, p_name: str = "", p_num_instances: int = 0, p_version:
    str = "", p_feature_space: MSpace = None, p_label_space: MSpace
    = None, p_sampler: Sampler = None, p_mode=0, p_logging=True,
    **p_kwargs)
```

Bases: [Mode](#), [Id](#), [ScientificObject](#)

Template class for data streams. Objects of this type can be used as iterators.

**Parameters**

- **p\_id** – Optional id of the stream. Default = None.
- **p\_name** (*str*) – Optional name of the stream. Default = ‘’.
- **p\_num\_instances** (*int*) – Optional number of instances in the stream. Default = 0.
- **p\_version** (*str*) – Optional version of the stream. Default = ‘’.
- **p\_feature\_space** ([MSpace](#)) – Optional feature space. Default = None.
- **p\_label\_space** ([MSpace](#)) – Optional label space. Default = None.
- **p\_sampler** – Optional sampler. Default: None.
- **p\_mode** – Operation mode. Default: Mode.C\_MODE\_SIM.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL.
- **p\_kwargs** (*dict*) – Further stream specific parameters.

**C\_TYPE** = 'Stream'

**get\_name**() → str

Returns the name of the stream.

**Returns**

**stream\_name** – Name of the stream.

**Return type**

str

**get\_url()** → str

Returns the URL of the scientific source/reference.

**Returns**

**url** – URL of the scientific source/reference.

**Return type**

str

**get\_num\_instances()** → int

Returns the number of instances of the stream.

**Returns**

**num\_inst** – Number of instances of the stream. If 0 the number is unknown.

**Return type**

int

**get\_feature\_space()** → *MSpace*

Returns the feature space of the stream.

**Returns**

**feature\_space** – Feature space of the stream.

**Return type**

*MSpace*

**\_setup\_feature\_space()** → *MSpace*

Custom method to set up the feature space of the stream. It is called by method `get_feature_space()`.

**Returns**

**feature\_space** – Feature space of the stream.

**Return type**

*MSpace*

**get\_label\_space()** → *MSpace*

Returns the label space of the stream.

**Returns**

**label\_space** – Label space of the stream.

**Return type**

*MSpace*

**\_setup\_label\_space()** → *MSpace*

Custom method to set up the label space of the stream. It is called by method `get_label_space()`.

**Returns**

**label\_space** – Label space of the stream.

**Return type**

*MSpace*

**set\_options(\*\*p\_kwargs)**

Method to set specific options for the stream. The possible options depend on the stream provider and stream itself.

**set\_random\_seed(p\_seed=None)**

Resets the internal random generator using the given seed.

**\_reset()**

Custom reset method for data stream. See method `__iter__()` for more details.

**setup\_sampler()** → *Sampler*

A static method to set up a sampler, which allows to set a sampler after instantiation of a stream.

**Returns**

An instantiated sampler.

**Return type**

*Sampler*

**\_get\_next()** → *Instance*

Custom method to determine the next data stream instance. At the end of the stream exception `StopIteration` is to be raised. See method `__next__()` for more details.

**Returns**

**instance** – Next instance of data stream or None.

**Return type**

*Instance*

**class** `mlpro.bf.streams.models.StreamProvider(p_logging=True)`

Bases: *Log*, *ScientificObject*

Template class for stream providers.

**Parameters**

**p\_logging** – Log level (see constants of class `Log`). Default: `Log.C_LOG_ALL`

**C\_TYPE** = 'Stream Provider'

**get\_stream\_list**(*p\_mode=0, p\_logging=True, \*\*p\_kwargs*) → list

Gets a list of provided streams by calling custom method `_get_stream_list()`.

**Parameters**

- **p\_mode** – Operation mode. Default: `Mode.C_MODE_SIM`.
- **p\_logging** – Log level of stream objects (see constants of class `Log`). Default: `Log.C_LOG_ALL`.
- **p\_kwargs** (*dict*) – Further stream specific parameters.

**Returns**

**stream\_list** – List of provided streams.

**Return type**

list

**\_get\_stream\_list**(*p\_mode=0, p\_logging=True, \*\*p\_kwargs*) → list

Custom method to get the list of provided streams. See method `get_stream_list()` for further details.

**Parameters**

- **p\_mode** – Operation mode. Default: `Mode.C_MODE_SIM`.
- **p\_logging** – Log level of stream objects (see constants of class `Log`). Default: `Log.C_LOG_ALL`.
- **p\_kwargs** (*dict*) – Further stream specific parameters.

**Returns**

**stream\_list** – List of provided streams.

**Return type**

list

**get\_stream**(*p\_id*: str = None, *p\_name*: str = None, *p\_mode*=0, *p\_logging*=True, *\*\*p\_kwargs*) → *Stream*Returns stream with the specified id by calling custom method `_get_stream()`.**Parameters**

- **p\_id** (str) – Optional Id of the requested stream. Default = None.
- **p\_name** (str) – Optional name of the requested stream. Default = None.
- **p\_mode** – Operation mode. Default: Mode.C\_MODE\_SIM.
- **p\_logging** – Log level of stream object (see constants of class Log). Default: Log.C\_LOG\_ALL.
- **p\_kwargs** (dict) – Further stream specific parameters.

**Returns**

s – Stream object or None in case of an error.

**Return type***Stream***\_get\_stream**(*p\_id*: str = None, *p\_name*: str = None, *p\_mode*=0, *p\_logging*=True, *\*\*p\_kwargs*) → *Stream*Custom method to get the specified stream. See method `get_stream()` for further details.**Parameters**

- **p\_id** (str) – Optional Id of the requested stream. Default = None.
- **p\_name** (str) – Optional name of the requested stream. Default = None.
- **p\_mode** – Operation mode. Default: Mode.C\_MODE\_SIM.
- **p\_logging** – Log level of stream object (see constants of class Log). Default: Log.C\_LOG\_ALL.
- **p\_kwargs** (dict) – Further stream specific parameters.

**Returns**

s – Stream object or None in case of an error.

**Return type***Stream*

```
class mlpro.bf.streams.models.StreamTask(p_name: str = None, p_range_max=1, p_duplicate_data: bool = False, p_visualize: bool = False, p_logging=True, **p_kwargs)
```

Bases: *Task*

Template class for stream-based tasks.

**Parameters**

- **p\_name** (str) – Optional name of the task. Default is None.
- **p\_range\_max** (int) – Maximum range of asynchronicity. See class Range. Default is Range.C\_RANGE\_PROCESS.
- **p\_duplicate\_data** (bool) – If True, instances will be duplicated before processing. Default = False.
- **p\_visualize** (bool) – Boolean switch for visualisation. Default = False.

- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL
- **p\_kwargs** (*dict*) – Further optional named parameters.

**C\_TYPE** = 'Stream-Task'

**C\_PLOT\_ACTIVE**: bool = True

**C\_PLOT\_STANDALONE**: bool = True

**C\_PLOT\_VALID\_VIEWS**: list = ['2D', '3D', 'ND']

**C\_PLOT\_DEFAULT\_VIEW**: str = 'ND'

**C\_PLOT\_ND\_XLABEL\_INST** = 'Instance index'

**C\_PLOT\_ND\_XLABEL\_TIME** = 'Time index'

**C\_PLOT\_ND\_YLABEL** = 'Feature Data'

**\_get\_custom\_run\_method**()

**run**(*p\_range*: int = None, *p\_wait*: bool = False, *p\_inst\_new*: List[Instance] = None, *p\_inst\_del*: List[Instance] = None)

Executes the specific actions of the task implemented in custom method `_run()`. At the end event `C_EVENT_FINISHED` is raised to start subsequent actions.

#### Parameters

- **p\_range** (*int*) – Optional deviating range of asynchronicity. See class Range. Default is None what means that the maximum range defined during instantiation is taken. Otherwise the minimum range of both is taken.
- **p\_wait** (*bool*) – If True, the method waits until all (a)synchronous tasks are finished.
- **p\_inst\_new** (*list*) – Optional list of new stream instances to be processed. If None, the list of the shared object is used instead. Default = None.
- **p\_inst\_del** (*list*) – List of obsolete stream instances to be removed. If None, the list of the shared object is used instead. Default = None.

**\_run\_wrapper**(*p\_inst\_new*: List[Instance], *p\_inst\_del*: List[Instance])

Internal use.

**\_run**(*p\_inst\_new*: List[Instance], *p\_inst\_del*: List[Instance])

Custom method that is called by method `run()`.

#### Parameters

- **p\_inst\_new** (*set*) – Set of new stream instances to be processed.
- **p\_inst\_del** (*set*) – Set of obsolete stream instances to be removed.

**init\_plot**(*p\_figure*: Figure = None, *p\_plot\_settings*: PlotSettings = None)

Initializes the plot functionalities of the class.

#### Parameters

- **p\_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p\_plot\_settings** (*PlotSettings*) – Optional plot settings. If None, the default view is plotted (see attribute `C_PLOT_DEFAULT_VIEW`).

**`_init_plot_2d`**(*p\_figure: Figure*, *p\_settings: PlotSettings*)

Default implementation for stream tasks. See class `mlpro.bf.plot.Plottable` for more details.

**`_init_plot_3d`**(*p\_figure: Figure*, *p\_settings: PlotSettings*)

Default implementation for stream tasks. See class `mlpro.bf.plot.Plottable` for more details.

**`_init_plot_nd`**(*p\_figure: Figure*, *p\_settings: PlotSettings*)

Default implementation for stream tasks. See class `mlpro.bf.plot.Plottable` for more details.

**`_finalize_plot_view`**(*p\_inst\_ref: Instance*)

**`update_plot`**(*p\_inst\_new: List[Instance] = None*, *p\_inst\_del: List[Instance] = None*, *\*\*p\_kwargs*)

Specialized definition of method `update_plot()` of class `mlpro.bf.plot.Plottable`.

#### Parameters

- **`p_inst_new`** (*List[Instance]*) – List of new stream instances to be plotted.
- **`p_inst_del`** (*List[Instance]*) – List of obsolete stream instances to be removed.
- **`p_kwargs`** (*dict*) – Further optional plot parameters.

**`_update_plot_2d`**(*p\_settings: PlotSettings*, *p\_inst\_new: list*, *p\_inst\_del: list*, *\*\*p\_kwargs*)

Default implementation for stream tasks. See class `mlpro.bf.plot.Plottable` for more details.

#### Parameters

- **`p_settings`** (*PlotSettings*) – Object with further plot settings.
- **`p_inst_new`** (*list*) – List of new stream instances to be plotted.
- **`p_inst_del`** (*list*) – List of obsolete stream instances to be removed.
- **`p_kwargs`** (*dict*) – Further optional plot parameters.

**`_update_plot_3d`**(*p\_settings: PlotSettings*, *p\_inst\_new: list*, *p\_inst\_del: list*, *\*\*p\_kwargs*)

Default implementation for stream tasks. See class `mlpro.bf.plot.Plottable` for more details.

#### Parameters

- **`p_settings`** (*PlotSettings*) – Object with further plot settings.
- **`p_inst_new`** (*list*) – List of new stream instances to be plotted.
- **`p_inst_del`** (*list*) – List of obsolete stream instances to be removed.
- **`p_kwargs`** (*dict*) – Further optional plot parameters.

**`_update_plot_nd`**(*p\_settings: PlotSettings*, *p\_inst\_new: list*, *p\_inst\_del: list*, *\*\*p\_kwargs*)

Default implementation for stream tasks. See class `mlpro.bf.plot.Plottable` for more details.

#### Parameters

- **`p_settings`** (*PlotSettings*) – Object with further plot settings.
- **`p_inst_new`** (*list*) – List of new stream instances to be plotted.
- **`p_inst_del`** (*list*) – List of obsolete stream instances to be removed.
- **`p_kwargs`** (*dict*) – Further optional plot parameters.

```
class mlpro.bf.streams.models.StreamWorkflow(p_name: str = None, p_range_max=1,  
                                             p_class_shared=<class  
                                             'mlpro.bf.streams.models.StreamShared'>, p_visualize:  
                                             bool = False, p_logging=True, **p_kwargs)
```

Bases: [StreamTask](#), [Workflow](#)

Workflow for stream processing. See class `bf.mt.Workflow` for further details.

#### Parameters

- **p\_name** (*str*) – Optional name of the task. Default is `None`.
- **p\_range\_max** (*int*) – Range of asynchronicity. See class `Range`. Default is `Range.C_RANGE_THREAD`.
- **p\_class\_shared** – Optional class for a shared object (class `StreamShared` or a child class of `StreamShared`). Default = `StreamShared`
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = `False`.
- **p\_logging** – Log level (see constants of class `Log`). Default: `Log.C_LOG_ALL`
- **p\_kwargs** (*dict*) – Further optional named parameters handed over to every task within.

**C\_TYPE** = 'Stream-Workflow'

**C\_PLOT\_ACTIVE**: `bool` = `True`

**run**(*p\_range*: `int` = `None`, *p\_wait*: `bool` = `False`, *p\_inst\_new*: `list` = `None`, *p\_inst\_del*: `list` = `None`)

Runs all stream tasks according to their predecessor relations.

#### Parameters

- **p\_range** (*int*) – Optional deviating range of asynchronicity. See class `Range`. Default is `None` what means that the maximum range defined during instantiation is taken. Otherwise the minimum range of both is taken.
- **p\_wait** (*bool*) – If `True`, the method waits until all (a)synchronous tasks are finished.
- **p\_inst\_new** (*list*) – Optional list of new stream instances to be processed. If `None`, the list of the shared object is used instead. Default = `None`.
- **p\_inst\_del** (*list*) – List of obsolete stream instances to be removed. If `None`, the list of the shared object is used instead. Default = `None`.

**init\_plot**(*p\_figure*: `Figure` = `None`, *p\_plot\_settings*: [PlotSettings](#) = `None`)

Initializes the plot of a workflow. The method creates a host figure for all tasks if no external host figure is parameterized. The sub-plots of the tasks are automatically arranged within the host figure.

See method `init_plot()` of class `mlpro.bf.plot.Plottable` for further details.

#### Parameters

- **p\_figure** (`Matplotlib.figure.Figure`, *optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is `None`.
- **p\_plot\_settings** ([PlotSettings](#)) – Optional plot settings. If `None`, the default view is plotted (see attribute `C_PLOT_DEFAULT_VIEW`).

**\_init\_plot\_2d**(*p\_figure*: `Figure`, *p\_settings*: [PlotSettings](#))

Default implementation for stream tasks. See class `mlpro.bf.plot.Plottable` for more details.

**\_init\_plot\_3d**(*p\_figure*: `Figure`, *p\_settings*: [PlotSettings](#))

Default implementation for stream tasks. See class `mlpro.bf.plot.Plottable` for more details.

**\_init\_plot\_nd**(*p\_figure*: `Figure`, *p\_settings*: [PlotSettings](#))

Default implementation for stream tasks. See class `mlpro.bf.plot.Plottable` for more details.

**update\_plot**(*p\_inst\_new*: list = None, *p\_inst\_del*: list = None, *\*\*p\_kwargs*)

Specialized definition of method `update_plot()` of class `mlpro.bf.plot.Plottable`.

**Parameters**

- **p\_inst\_new** (*list*) – List of new stream instances to be plotted.
- **p\_inst\_del** (*list*) – List of obsolete stream instances to be removed.
- **p\_kwargs** (*dict*) – Further optional plot parameters.

**class** `mlpro.bf.streams.models.StreamScenario`(*p\_mode*, *p\_cycle\_limit*=0, *p\_visualize*: bool = False, *p\_logging*=True)

Bases: `ScenarioBase`

Template class for stream based scenarios.

**Parameters**

- **p\_mode** – Operation mode. See `Mode.C_VALID_MODES` for valid values. Default = `Mode.C_MODE_SIM`.
- **p\_cycle\_limit** (*int*) – Maximum number of cycles. Default = 0 (no limit).
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class `Log`). Default: `Log.C_LOG_ALL`.

**C\_TYPE** = 'Stream-Scenario'

**C\_PLOT\_ACTIVE**: bool = True

**setup**()

Specialized method to set up a stream scenario. It is automatically called by the constructor and calls in turn the custom method `_setup()`.

**\_setup**(*p\_mode*, *p\_visualize*: bool, *p\_logging*)

Custom method to set up a stream scenario consisting of a stream and a processing stream workflow.

**Parameters**

- **p\_mode** – Operation mode. See `Mode.C_VALID_MODES` for valid values. Default = `Mode.C_MODE_SIM`.
- **p\_visualize** (*bool*) – Boolean switch for visualisation.
- **p\_logging** – Log level (see constants of class `Log`). Default: `Log.C_LOG_ALL`.

**Returns**

- **stream** (*Stream*) – A stream object.
- **workflow** (*StreamWorkflow*) – A stream workflow object.

**\_set\_mode**(*p\_mode*)

Custom method to set the operation mode of components of the scenario. See method `set_mode()` for further details.



## Parameter

### **p\_mode**

Operation mode. See class `bf.ops.Mode` for further details.

### **\_reset(p\_seed)**

Custom method to reset the components of the scenario and to set the given random seed value. See method `reset()` for further details.

#### Parameters

**p\_seed** (*int*) – Seed value for internal random generator

### **get\_latency()** → *timedelta*

Returns the latency of the scenario. To be implemented in child class.

### **\_run\_cycle()**

Gets next instance from the stream and lets process it by the stream workflow.

#### Returns

- **success** (*bool*) – True on success. False otherwise.
- **error** (*bool*) – True on error. False otherwise.
- **adapted** (*bool*) – True, if something within the scenario has adapted something in this cycle. False otherwise.
- **end\_of\_data** (*bool*) – True, if the end of the related data source has been reached. False otherwise.

### **\_init\_figure()** → *Figure*

Custom method to initialize a suitable standalone Matplotlib figure.

#### Returns

**figure** – Matplotlib figure object to host the subplot(s)

#### Return type

`Matplotlib.figure.Figure`

### **init\_plot(p\_figure: *Figure* = None, p\_plot\_settings: *PlotSettings* = None)**

Initializes the plot functionalities of the class.

#### Parameters

- **p\_figure** (*Matplotlib.figure.Figure*, *optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is `None`.
- **p\_plot\_settings** (*PlotSettings*) – Optional plot settings. If `None`, the default view is plotted (see attribute `C_PLOT_DEFAULT_VIEW`).

### **update\_plot(\*\*p\_kwargs)**

Plot updates take place during workflow/task processing and are disabled here...

### **get\_stream()** → *Stream*

### **get\_workflow()** → *StreamWorkflow*



## BF-PHYSICS - Physics

## Legend:



In progress



In validation



Ready to use

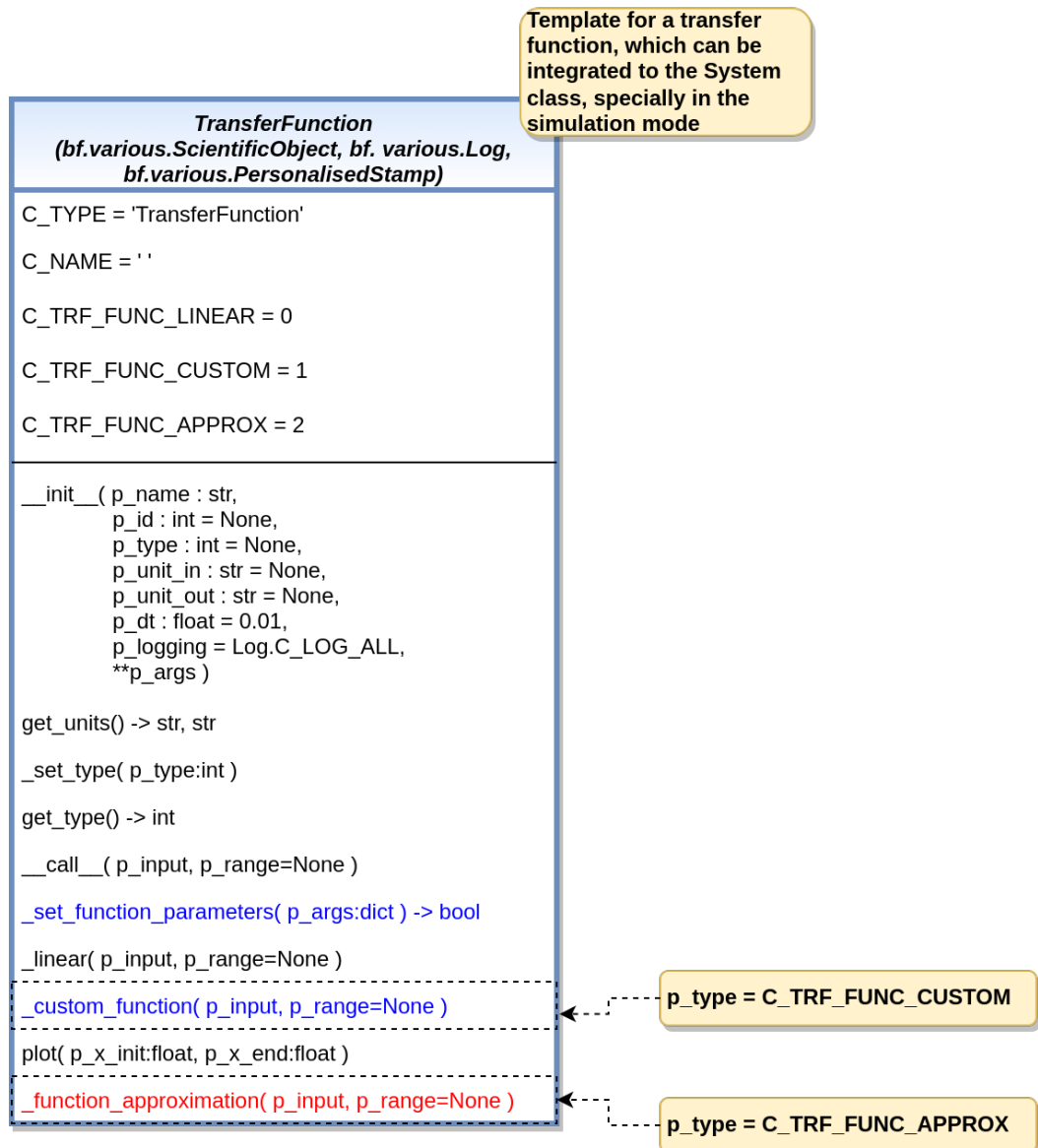


Depends on



Inherits from

`attribute / method()`    New public attribute/method to be implemented  
`attribute / method()`    New/inherited public attribute/method that is implemented here  
`attribute / method()`    New protected method/attribute to be implemented  
`attribute / method()`    New/inherited protected attribute/method that is implemented here



This module provides models and templates for physics.

```
class mlpro.bf.physics.basics.TransferFunction(p_name: str, p_id: int = None, p_type: int = None,
                                              p_unit_in: str = None, p_unit_out: str = None, p_dt:
                                              float = 0.01, p_logging=True, **p_args)
```

Bases: *ScientificObject*, *Log*, *PersonalisedStamp*

This class serves as a base class of transfer functions, which provides the main attributes of a transfer function. By default, there are several ready-to-use transfer function types available. If none of them suits to your transfer function, then you can also select a 'custom' type of transfer function and design your own function. Another possibility is to use a function approximation functionality provided by MLPro (coming soon).

#### Parameters

- **p\_name** (*str*) – name of the transfer function.
- **p\_id** (*int*) – unique id of the transfer function. Default: None.
- **p\_type** (*int*) – type of the transfer function. Default: None.
- **p\_unit\_in** (*str*) – unit of the transfer function's input. Default: None.
- **p\_unit\_out** (*str*) – unit of the transfer function's output. Default: None.
- **p\_dt** (*float*) – delta time. Default: 0.01.
- **p\_logging** – Log level (see constants of class *Log*). Default: *Log.C\_LOG\_ALL*.
- **p\_args** (*dict*) – extra parameter for each specific transfer function.

#### C\_TYPE

type of the base class. Default: 'TransferFunction'.

##### Type

str

#### C\_NAME

name of the transfer function. Default: ''.

##### Type

str

#### C\_TRF\_FUNC\_LINEAR

linear function. Default: 0.

##### Type

int

#### C\_TRF\_FUNC\_CUSTOM

custom transfer function. Default: 1.

##### Type

int

#### C\_TRF\_FUNC\_APPROX

function approximation. Default: 2.

##### Type

int

**C\_TYPE = 'TransferFunction'**

**C\_TRF\_FUNC\_LINEAR = 0**

**C\_TRF\_FUNC\_CUSTOM** = 1

**C\_TRF\_FUNC\_APPROX** = 2

**C\_NAME** = ''

**get\_units()**

This method provides a functionality to get the SI units of the input and output data.

**Returns**

- **self.\_unit\_in** (*str*) – the SI unit of the input data.
- **self.\_unit\_out** (*str*) – the SI unit of the output data.

**\_set\_type**(*p\_type: int*)

This method provides a functionality to set the type of the transfer function.

**Parameters**

**p\_type** (*int*) – the type of the transfer function.

**get\_type**() → int

This method provides a functionality to get the type of the transfer function.

**Returns**

the type of the transfer function.

**Return type**

int

**\_set\_function\_parameters**(*p\_args: dict*) → bool

This method provides a functionality to set the parameters of the transfer function.

**Parameters**

**p\_args** (*dict*) – set of parameters of the transfer function.

**Returns**

true means no parameters are missing.

**Return type**

bool

**\_linear**(*p\_input: float, p\_range=None*) → float

This method provides a functionality for linear transfer function.

Formula →  $y = mx + b$   $y = \text{output}$   $m = \text{slope}$   $x = \text{input}$   $b = y\text{-intercept}$

**Parameters**

- **p\_input** (*float*) – input value.
- **p\_range** – range of the calculation. None means 0. Default: None.

**Returns**

output value.

**Return type**

float

**\_custom\_function**(*p\_input, p\_range=None*)

This function represents the template to create a custom function and must be redefined.

**Parameters**

- **p\_input** – input value.
- **p\_range** – range of the calculation. None means 0. Default: None.

**Returns**

output value.

**Return type**

float

**plot**(*p\_x\_init: float, p\_x\_end: float*)

This methods provides functionality to plot the defined function within a range.

**Parameters**

- **p\_x\_init** (*float*) – The initial value of the input (x-axis).
- **p\_x\_end** (*float*) – The end value of the input (x-axis).

**\_function\_approximation**(*p\_input, p\_range=None*)

The function approximation is not yet ready (coming soon).

**Parameters**

**p\_input** (*TYPE*) – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

bool

BF-PHYSICS-UNITCONVERTER - Unit Converters

<i>UnitConverter</i> (bf.physics.TransferFunction)
C_TYPE = 'UnitConverter'
C_NAME = ''
C_UNIT_CONV_LENGTH = 0
C_UNIT_CONV_PRESSURE = 1
C_UNIT_CONV_CURRENT = 2
C_UNIT_CONV_FORCE = 3
C_UNIT_CONV_POWER = 4
C_UNIT_CONV_MASS = 5
C_UNIT_CONV_TIME = 6
C_UNIT_CONV_TEMPERATURE = 7
 __init__( p_name : str, p_id : int = None, p_type : int = None, p_unit_in : str = None, p_unit_out : str = None, p_dt : float = 0.01, p_logging = Log.C_LOG_ALL, **p_args )  __call__( p_input, p_range=None )  _set_function_parameters( p_args=None ) -> bool  _scalar_conversion( p_input: float ) -> float  _temperature( p_input: float ) -> float

Legend:

In progress

In validation

Ready to use

Depends on

Inherits from

attribute / method()

New public attribute/method to be implemented

attribute / method()

New/inherited public attribute/method that is implemented here

attribute / \_method()

New protected method/attribute to be implemented

attribute / \_method()

New/inherited protected Method that is implemented here

This module provides models for unit conversions.

```
class mlpro.bf.physics.unitconverter.UnitConverter(p_name: str, p_id: int = None, p_type: int =  
None, p_unit_in: str = None, p_unit_out: str =  
None, p_logging=True, **p_args)
```

Bases: [\*TransferFunction\*](#)

This class serves as a base class of unit converters, which inherits the main attributes from a transfer function. By default, there are several ready-to-use unit converters available, such as, Length, Temperature, Pressure, Electric Current, Force, Power, Mass, and Time.

#### Parameters

- **p\_name** (*str*) – name of the unit converter.
- **p\_id** (*int*) – unique id of the unit converter. Default: None.
- **p\_type** (*int*) – type of the unit converter. Default: None.
- **p\_unit\_in** (*str*) – unit of the unit converter’s input. Default: None.
- **p\_unit\_out** (*str*) – unit of the unit converter’s output. Default: None.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL.

#### C\_TYPE

type of the base class. Default: ‘UnitConverter’.

##### Type

str

#### C\_NAME

name of the unit converter. Default: ‘’.

##### Type

str

#### C\_UNIT\_CONV\_LENGTH

unit converter for length. Default: 0.

##### Type

int

#### C\_UNIT\_CONV\_PRESSURE

unit converter for pressure. Default: 1.

##### Type

int

#### C\_UNIT\_CONV\_CURRENT

unit converter for electric current. Default: 2.

##### Type

int

#### C\_UNIT\_CONV\_FORCE

unit converter for force. Default: 3.

##### Type

int



**C\_UNIT\_CONV\_POWER**

unit converter for power. Default: 4.

**Type**

int

**C\_UNIT\_CONV\_MASS**

unit converter for mass. Default: 5.

**Type**

int

**C\_UNIT\_CONV\_TIME**

unit converter for time. Default: 6.

**Type**

int

**C\_UNIT\_CONV\_TEMPERATURE**

unit converter for temperature. Default: 7.

**Type**

int

**C\_TYPE** = 'UnitConverter'

**C\_NAME** = ''

**C\_UNIT\_CONV\_LENGTH** = 0

**C\_UNIT\_CONV\_PRESSURE** = 1

**C\_UNIT\_CONV\_CURRENT** = 2

**C\_UNIT\_CONV\_FORCE** = 3

**C\_UNIT\_CONV\_POWER** = 4

**C\_UNIT\_CONV\_MASS** = 5

**C\_UNIT\_CONV\_TIME** = 6

**C\_UNIT\_CONV\_TEMPERATURE** = 7

**\_set\_function\_parameters**(*p\_args=None*) → bool

This method provides a functionality to set the parameters of the unit converter.

**Parameters**

**p\_args** – not necessary for a unit converter. Default: None.

**Returns**

true means initialization is successful.

**Return type**

bool

**\_scalar\_conversion**(*p\_input: float*) → float

This method provides a scalar conversion functionality.

**Parameters**

**p\_input** (*float*) – input value.

**Returns**

output value.

**Return type**

float

**\_temperature**(*p\_input: float*) → float

This method provides a temperature conversion functionality.

**Parameters**

**p\_input** (*float*) – input value.

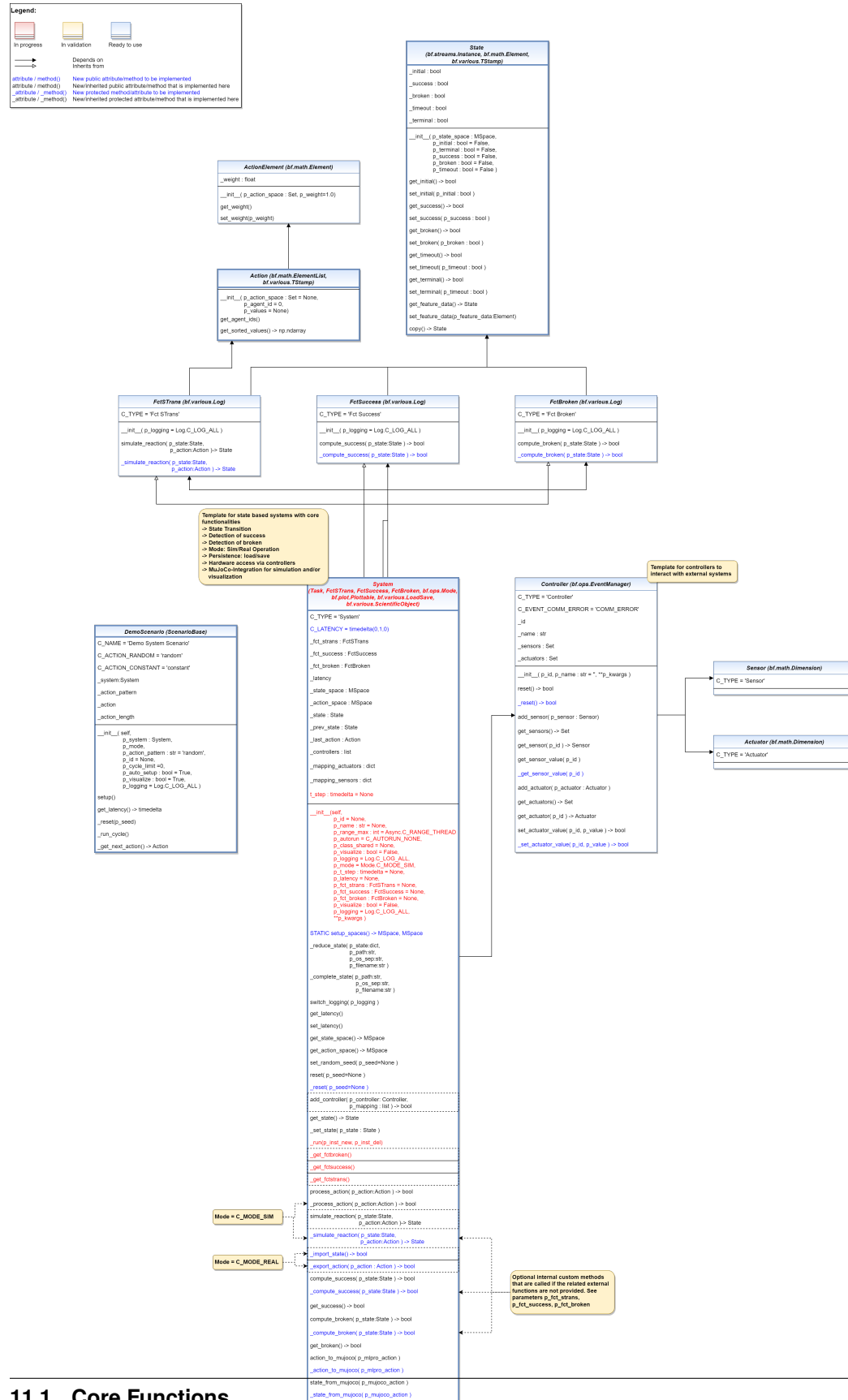
**Returns**

output value.

**Return type**

float

## BF-SYSTEMS - State-based Systems



This module provides models and templates for state based systems.

```
class mlpro.bf.systems.basics.State(p_state_space: MSpace, p_initial: bool = False, p_terminal: bool = False, p_success: bool = False, p_broken: bool = False, p_timeout: bool = False, **p_kwargs)
```

Bases: *Instance*, *Element*, *TStamp*

State of a system as an element of a given state space. Additionally, the state can be labeled with various properties.

#### Parameters

- **p\_state\_space** (*MSpace*) – State space of the related system.
- **p\_initial** (*bool*) – This optional flag signals that the state is the first one after a reset. Default=False.
- **p\_terminal** (*bool*) – This optional flag labels the state as a terminal state. Default=False.
- **p\_success** (*bool*) – This optional flag labels the state as an objective state. Default=False.
- **p\_broken** (*bool*) – This optional flag labels the state as a final error state. Default=False.
- **p\_timeout** (*bool*) – This optional flag signals that the cycle limit of an episode has been reached. Default=False.

**get\_initial()** → *bool*

**set\_initial**(*p\_initial: bool*)

**get\_success()** → *bool*

**set\_success**(*p\_success: bool*)

**get\_broken()** → *bool*

**set\_broken**(*p\_broken: bool*)

**get\_timeout()** → *bool*

**set\_timeout**(*p\_timeout: bool*)

**get\_terminal()** → *bool*

**set\_terminal**(*p\_terminal: bool*)

**get\_feature\_data()** → *Element*

**set\_feature\_data**(*p\_feature\_data: Element*)

**copy()**

Returns a copy of the state element

#### Returns

**copied\_state** – The copy of original state object.

#### Return type

*State*

```
class mlpro.bf.systems.basics.ActionElement(p_action_space: Set, p_weight: float = 1.0)
```

Bases: *Element*

Single entry of an action. See class Action for further details.

**Parameters**

- **p\_action\_space** (*Set*) – Related action space.
- **p\_weight** (*float*) – Weight of action element. Default = 1.0.

**get\_weight()**

**set\_weight**(*p\_weight*)

**class** mlpro.bf.systems.basics.**Action**(*p\_agent\_id=0, p\_action\_space: Set = None, p\_values: ndarray = None*)

Bases: *ElementList, TStamp*

Objects of this class represent actions of (multi-)agents. Every element of the internal list is related to an agent, and its partial subsection. Action values for the first agent can be added while object instantiation. Action values of further agents can be added by using method `self.add_elem()`.

**Parameters**

- **p\_agent\_id** – Unique id of (first) agent to be added
- **p\_action\_space** (*Set*) – Action space of (first) agent to be added
- **p\_values** (*np.ndarray*) – Action values of (first) agent to be added

**get\_agent\_ids()**

**get\_sorted\_values()** → *ndarray*

**class** mlpro.bf.systems.basics.**FctSTrans**(*p\_logging=True*)

Bases: *Log*

Template class for state transition functions.

**Parameters**

**p\_logging** – Log level (see class *Log* for more details). Default = *Log.C\_LOG\_ALL*.

**C\_TYPE** = 'Fct STrans'

**simulate\_reaction**(*p\_state: State, p\_action: Action, p\_t\_step: timedelta = None*) → *State*

Simulates a state transition based on a state and action. Custom method `_simulate_reaction()` is called.

**Parameters**

- **p\_state** (*State*) – System state.
- **p\_action** (*Action*) – Action to be processed.

**Returns**

**new\_state** – Result state after state transition.

**Return type**

*State*

**\_simulate\_reaction**(*p\_state: State, p\_action: Action, p\_t\_step: timedelta = None*) → *State*

Custom method for a simulated state transition. See method `simulate_reaction()` for further details.

**class** mlpro.bf.systems.basics.**FctSuccess**(*p\_logging=True*)

Bases: *Log*

Template class for functions that determine whether or not a state is a success state.

**Parameters**

**p\_logging** – Log level (see class *Log* for more details). Default = *Log.C\_LOG\_ALL*.

**C\_TYPE = 'Fct Success'**

**compute\_success**(*p\_state*: [State](#)) → bool

Assesses the given state regarding success criteria. Custom method `_compute_success()` is called.

**Parameters**

**p\_state** ([State](#)) – System state.

**Returns**

**success** – True, if given state is a success state. False otherwise.

**Return type**

bool

**\_compute\_success**(*p\_state*: [State](#)) → bool

Custom method for assessment for success. See method `compute_success()` for further details.

**class** `mlpro.bf.systems.basics.FctBroken`(*p\_logging*=*True*)

Bases: [Log](#)

Template class for functions that determine whether or not a state is a broken state.

**Parameters**

**p\_logging** – Log level (see class `Log` for more details). Default = `Log.C_LOG_ALL`.

**C\_TYPE = 'Fct Broken'**

**compute\_broken**(*p\_state*: [State](#)) → bool

Assesses the given state regarding breakdown criteria. Custom method `_compute_success()` is called.

**Parameters**

**p\_state** ([State](#)) – System state.

**Returns**

**broken** – True, if given state is a breakdown state. False otherwise.

**Return type**

bool

**\_compute\_broken**(*p\_state*: [State](#)) → bool

Custom method for assessment for breakdown. See method `compute_broken()` for further details.

**class** `mlpro.bf.systems.basics.Sensor`(*p\_name\_short*, *p\_base\_set*='R', *p\_name\_long*="", *p\_name\_latex*="",  
*p\_unit*="", *p\_unit\_latex*="", *p\_boundaries*: *list* = [], *p\_description*="",  
*p\_symmetrical*: *bool* = *False*, *p\_logging*=*False*, *\*\*p\_kwargs*)

Bases: [Dimension](#)

Template for a sensor.

**C\_TYPE = 'Sensor'**

**class** `mlpro.bf.systems.basics.Actuator`(*p\_name\_short*, *p\_base\_set*='R', *p\_name\_long*="",  
*p\_name\_latex*="", *p\_unit*="", *p\_unit\_latex*="", *p\_boundaries*: *list* =  
[], *p\_description*="", *p\_symmetrical*: *bool* = *False*,  
*p\_logging*=*False*, *\*\*p\_kwargs*)

Bases: [Dimension](#)

Template for an actuator.

**C\_TYPE = 'Actuator'**

**class** mlpro.bf.systems.basics.**Controller**(*p\_id*, *p\_name*: *str* = "", *p\_logging*: *bool* = *True*, *\*\*p\_kwargs*)

Bases: [EventManager](#)

Template for a controller that enables access to sensors and actuators.

#### Parameters

- **p\_id** – Unique id of the controller.
- **p\_name** (*str*) – Optional name of the controller.
- **p\_logging** – Log level (see class Log for more details). Default = Log.C\_LOG\_ALL.
- **p\_kwargs** (*dict*) – Further keyword arguments specific to the controller.

**C\_EVENT\_COMM\_ERROR**

Event that is raised on a communication error

**C\_TYPE** = 'Controller'

**C\_EVENT\_COMM\_ERROR** = 'COMM\_ERROR'

**reset()** → *bool*

Resets the controller by calling custom method `_reset()`.

#### Returns

**result** – True, if successful. False otherwise. Additionally event **C\_EVENT\_COMM\_ERROR** is raised.

#### Return type

*bool*

**\_reset()** → *bool*

Custom reset method.

#### Returns

**result** – True, if successful. False otherwise.

#### Return type

*bool*

**add\_sensor**(*p\_sensor*: [Sensor](#))

Adds a sensor to the controller.

#### Parameters

**p\_sensor** ([Sensor](#)) – Sensor object to be added.

**get\_sensors()** → [Set](#)

Returns the internal set of sensors.

#### Returns

**sensors** – Set of sensors.

#### Return type

[Set](#)

**get\_sensor**(*p\_id*) → [Sensor](#)

Returns a sensor.

**get\_sensor\_value**(*p\_id*)

Determines the value of a sensor by calling custom method `_get_sensor_value()`.

**Parameters**

**p\_id** – Id of the sensor.

**Returns**

Current value of the sensor or None on a communication error. In that case, event C\_EVENT\_COMM\_ERROR is raised additionally.

**Return type**

value

**\_get\_sensor\_value(p\_id)**

Custom method to get a sensor value. See method get\_sensor\_value() for further details.

**Parameters**

**p\_id** – Id of the sensor.

**Returns**

Current value of the sensor or None on a communication error.

**Return type**

value

**add\_actuator(p\_actuator: [Actuator](#))**

Adds an actuator to the controller.

**Parameters**

**p\_actuator** ([Actuator](#)) – Actuator object to be added.

**get\_actuators()** → [Set](#)

Returns the internal set of actuators.

**Returns**

**actuators** – Set of actuators.

**Return type**

[Set](#)

**get\_actuator(p\_id)** → [Actuator](#)

Returns an actuator.

**set\_actuator\_value(p\_id, p\_value)** → bool

Sets the value of an actuator by calling custom method \_set\_actuator\_value().

**Parameters**

- **p\_id** – Id of the actuator.
- **p\_value** – New actuator value.

**Returns**

**successful** – True, if successful. False otherwise. In that case, event C\_EVENT\_COMM\_ERROR is raised additionally.

**Return type**

bool

**\_set\_actuator\_value(p\_id, p\_value)** → bool

Custom method to set an actuator value. See method set\_sensor\_value() for further details.

**Parameters**

- **p\_id** – Id of the actuator.
- **p\_value** – New actuator value.



**Returns**

**successful** – True, if successful. False otherwise.

**Return type**

bool

**class** mlpro.bf.systems.basics.**SystemShared**(*p\_range: int = 0*)

Bases: *Shared*

A specialised shared object for managing IPC between MultiSystems.

**TODO**

Entry Systems to be handled yet, that get action from outside.

**Parameters**

**p\_range** – The multiprocessing range for the specific process. Default is None.

**\_spaces**

Spaces of all the systems registered in the Shared Object.

**\_states**

States of all the systems registered in the Shared Object.

**\_actions**

Corresponding Actions for all the systems.

**\_action\_dimensions**

All the dimensions present in the Shared Object.

**\_mappings**

Mapping configurations for system to action mapping.

**C\_NAME** = 'System Shared'

**reset**(*p\_seed: int = None*)

Resets the shared object.

**Parameters**

**p\_seed** (*int*) – Seed for reproducibility.

**update\_state**(*p\_sys\_id, p\_state: State*) → bool

Updates the states in the Shared Object.

**Parameters**

**p\_state** (*State*) – The id of the system, for which action is to be fetched.

**Return type**

bool

**\_map\_values**(*p\_state: State = None, p\_action: Action = None*)

Updates the action values based on a new state, in a MultiSystem Context.

**Parameters**

- **p\_sys\_id** – Id of the system from which the state is received.
- **p\_state** (*State*) – The State of the system which affects the action.

**get\_actions**()

**get\_action**(*p\_sys\_id*) → *Action*

Fetches the corresponding action for a particular system.

**Parameters**

**p\_sys\_id** – The id of the system, for which action is to be fetched.

**Returns**

**action** – The corresponding action for the system.

**Return type**

*Action*

**get\_states**()

Fetch the states of all the internal systems

**Returns**

**states** – Returns the state of each of the system registered on the shared object.

**Return type**

dict

**get\_state**(*p\_sys\_id*) → *State*

Fetches the state of a particular system from the Shared Object.

**Parameters**

**p\_sys\_id** – The id of the system, of which state is to be fetched.

**Returns**

**state** – The corresponding state of the system.

**Return type**

*State*

**\_map**(*p\_input\_dim=None*)

Maps a dimension to output dimension with info about output sys, output dim type and output dim.

**Parameters**

- **p\_sys\_id** – Id of the system for which action is to be mapped into the mappings.
- **p\_state** (*State*) – The State to be mapped into the ‘states’ dictionary.

**Returns**

**mapping** – A tuple of tuples of System id and dimension id mappings from State to Action respectively.

**Return type**

(output\_sys, output\_dim\_type, output\_dim)

**register\_system**(*p\_sys\_id=None*, *p\_state\_space: MSpace = None*, *p\_action\_space: MSpace = None*, *p\_mappings=None*)

Registers the system in the Shared Object and sets up the dimension to dimension mapping.

**Parameters**

- **p\_system** (*System*) – The system to be registered.
- **p\_mappings** – Mappings corresponding the system in the form: ( (ip dim\_type, op dim\_type), (input\_sys\_id, input\_dim\_id) , (op\_sys\_id, op\_dimension\_id) )

```
class mlpro.bf.systems.basics.System(p_id=None, p_name: str = None, p_range_max: int = 0,
                                     p_autorun=0, p_class_shared=None, p_mode=0, p_latency:
                                     timedelta = None, p_t_step: timedelta = None, p_fct_strans:
                                     FctSTrans = None, p_fct_success: FctSuccess = None,
                                     p_fct_broken: FctBroken = None, p_mujoco_file=None,
                                     p_frame_skip: int = 1, p_state_mapping=None,
                                     p_action_mapping=None, p_camera_conf: tuple = (None, None,
                                     None), p_visualize: bool = False, p_logging=True, **p_kwargs)
```

Bases: *FctSTrans*, *FctSuccess*, *FctBroken*, *Task*, *Mode*, *Plottable*, *Persistent*, *ScientificObject*

Base class for state based systems.

#### Parameters

- **p\_mode** – Mode of the system. Possible values are Mode.C\_MODE\_SIM(default) or Mode.C\_MODE\_REAL.
- **p\_latency** (*timedelta*) – Optional latency of the system. If not provided, the internal value of constant C\_LATENCY is used by default.
- **p\_fct\_strans** (*FctSTrans*) – Optional external function for state transition.
- **p\_fct\_success** (*FctSuccess*) – Optional external function for state evaluation ‘success’.
- **p\_fct\_broken** (*FctBroken*) – Optional external function for state evaluation ‘broken’.
- **p\_mujoco\_file** – Path to XML file for MuJoCo model.
- **p\_frame\_skip** (*int*) – Frame to be skipped every step. Default = 1.
- **p\_state\_mapping** – State mapping if the MLPro state and MuJoCo state have different naming.
- **p\_action\_mapping** – Action mapping if the MLPro action and MuJoCo action have different naming.
- **p\_use\_radian** (*bool*) – Use radian if the action and the state based on radian unit. Default = True.
- **p\_camera\_conf** (*tuple*) – Default camera configuration on MuJoCo Simulation (xyz position, elevation, distance).
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see class Log for more details). Default = Log.C\_LOG\_ALL.

#### **\_latency**

Latency of the system.

##### Type

*timedelta*

#### **\_state**

Current state of system.

##### Type

*State*

#### **\_prev\_state**

Previous state of system.

##### Type

*State*

**\_last\_action**

Last action.

**Type**

*Action*

**\_fct\_strans**

Internal state transition function.

**Type**

*FctSTrans*

**\_fct\_success**

Internal function for state evaluation 'success'.

**Type**

*FctSuccess*

**\_fct\_broken**

Internal function for state evaluation 'broken'.

**Type**

*FctBroken*

**C\_TYPE = 'System'**

**C\_LATENCY = datetime.timedelta(seconds=1)**

**C\_PLOT\_ACTIVE: bool = True**

**static setup\_spaces()**

Static template method to set up and return state and action space of environment.

**Returns**

- **state\_space** (*MSpace*) – State space object
- **action\_space** (*MSpace*) – Action space object

**\_reduce\_state**(*p\_state: dict, p\_path: str, p\_os\_sep: str, p\_filename\_stub: str*)

An embedded MuJoCo system can not be pickled and needs to be removed from the pickle stream.

**\_complete\_state**(*p\_path: str, p\_os\_sep: str, p\_filename\_stub: str*)

Custom method to complete the object state (=self) from external data sources. This method is called by standard method `__setstate__()` during unpickling the object from an external file.

**Parameters**

- **p\_path** (*str*) – Path of the object pickle file (and further optional related files)
- **p\_os\_sep** (*str*) – OS-specific path separator.
- **p\_filename\_stub** (*str*) – Filename stub to be used for further optional custom data files

**switch\_logging**(*p\_logging*)

Sets new log level.

**Parameters**

**p\_logging** – Log level (constant `C_LOG_LEVELS` contains valid values)

**get\_latency()** → *timedelta*

Returns latency of the system.

**set\_latency**(*p\_latency: timedelta = None*) → None

Sets latency of the system. If *p\_latency* is None latency will be reset to internal value of attribute C\_LATENCY.

**Parameters**

**p\_latency** (*timedelta*) – New latency value

**get\_state\_space**() → *MSpace*

**get\_action\_space**() → *MSpace*

**get\_fct\_strans**()

Returns the state transition function of the system, if exists, otherwise, the system itself.

**Returns**

**fct\_strans** – State transition function of the system, if exists. Otherwise, system itself.

**Return type**

*FctSTrans*

**get\_fct\_broken**()

Returns the broken computation function of the system, if exists, otherwise, the system itself.

**Returns**

**fct\_broken** – Broken computation function of the system, if exists. Otherwise, system itself.

**Return type**

*FctBroken*

**get\_fct\_success**()

Returns the Success computation function of the system, if exists, otherwise, the system itself.

**Returns**

**fct\_success** – Success computation function of the system, if exists. Otherwise, system itself.

**Return type**

*FctSuccess*

**set\_random\_seed**(*p\_seed=None*)

Resets the internal random generator using the given seed.

**Parameters**

**p\_seed** (*int*) – Seed parameter for an internal random generator

**reset**(*p\_seed=None*) → None

Resets the system to an initial state. If MuJoCo is not used, the custom method `_reset()` is called.

**Parameters**

**p\_seed** (*int*) – Seed parameter for an internal random generator

**\_reset**(*p\_seed=None*) → None

Custom method to reset the system to an initial/defined state. Use method `_set_status()` to set the state.

**Parameters**

**p\_seed** (*int*) – Seed parameter for an internal random generator

**add\_controller**(*p\_controller: Controller, p\_mapping: list*) → bool

Adds a controller and a related mapping of states and actions to sensors and actuators.

**Parameters**

- **p\_controller** (*Controller*) – Controller object to be added.

- **p\_mapping** (*list*) – A list of mapping tuples following the syntax ( [Type = ‘S’ or ‘A’], [Name of state/action] [Name of sensor/actuator] )

**Returns**

**successful** – True, if controller and related mapping was added successfully. False otherwise.

**Return type**

bool

**get\_state()** → *State*

Returns current state of the system.

**\_set\_state**(*p\_state: State*)

Explicitly sets the current state of the system. Internal use only.

**get\_so()** → *SystemShared*

Returns the associated shared object.

**Returns**

**so** – Shared object of type Shared (or inherited)

**Return type**

*Shared*

**\_run**(*p\_action: Action*, *p\_t\_step: timedelta = None*)

Run method that runs the system as a task. It runs the process\_action() method of the system with action as a parameter.

**Parameters**

**p\_t\_step** (*timedelta*) – Time for which the system must be simulated.

**process\_action**(*p\_action: Action*, *p\_t\_step: timedelta = None*) → bool

Processes a state transition based on the current state and a given action. The state transition itself is implemented in child classes in the custom method \_process\_action().

**Parameters**

- **p\_action** (*Action*) – Action to be processed
- **p\_t\_step** (*timedelta*) – The timestep for which the system is to be simulated

**Returns**

**success** – True, if action processing was successfull. False otherwise.

**Return type**

bool

**\_process\_action**(*p\_action: Action*, *p\_t\_step: timedelta = None*) → bool

Internal custom method for state transition with default implementation. To be redefined in a child class on demand. See method process\_action() for further details.

**simulate\_reaction**(*p\_state: State = None*, *p\_action: Action = None*, *p\_t\_step: timedelta = None*) → *State*

Simulates a state transition based on a state and an action. The simulation step itself is carried out either by an internal custom implementation in method \_simulate\_reaction() or by an embedded external function.

**Parameters**

- **p\_state** (*State*) – Current state.
- **p\_action** (*Action*) – Action.

**Returns**

Subsequent state after transition

**Return type***State***\_simulate\_reaction**(*p\_state*: *State*, *p\_action*: *Action*, *p\_step*: *timedelta = None*) → *State*

Custom method for a simulated state transition. Implement this method if no external state transition function is used. See method `simulate_reaction()` for further details.

**action\_to\_mujoco**(*p\_mlpro\_action*)

Action conversion method from converting MLPro action to MuJoCo action.

**\_action\_to\_mujoco**(*p\_mlpro\_action*)

Custom method for to do transition between MuJoCo state and MLPro state. Implement this method if the MLPro state has different dimension from MuJoCo state.

**Parameters****p\_mujoco\_state** (*Numpy*) – MLPro action.**Returns**

Modified MLPro action

**Return type***Numpy***state\_from\_mujoco**(*p\_mujoco\_state*)

State conversion method from converting MuJoCo state to MLPro state.

**\_state\_from\_mujoco**(*p\_mujoco\_state*)

Custom method for to do transition between MuJoCo state and MLPro state. Implement this method if the MLPro state has different dimension from MuJoCo state.

**Parameters****p\_mujoco\_state** (*Numpy*) – MuJoCo state.**Returns**

Modified MuJoCo state

**Return type***Numpy***\_import\_state**() → bool**\_export\_action**(*p\_action*: *Action*) → bool**compute\_success**(*p\_state*: *State*) → bool

Assesses the given state whether it is a ‘success’ state. Assessment is carried out either by a custom implementation in method `_compute_success()` or by an embedded external function.

**Parameters****p\_state** (*State*) – State to be assessed.**Returns****success** – True, if the given state is a ‘success’ state. False otherwise.**Return type**

bool

**\_compute\_success**(*p\_state*: *State*) → bool

Custom method for assessment for success. Implement this method if no external function is used. See method `compute_success()` for further details.

**get\_success()** → bool

**compute\_broken**(*p\_state*: [State](#)) → bool

Assesses the given state whether it is a 'broken' state. Assessment is carried out either by a custom implementation in method `_compute_broken()` or by an embedded external function.

**Parameters**

**p\_state** ([State](#)) – State to be assessed.

**Returns**

**broken** – True, if the given state is a 'broken' state. False otherwise.

**Return type**

bool

**\_compute\_broken**(*p\_state*: [State](#)) → bool

Custom method for assessment for breakdown. Implement this method if no external function is used. See method `compute_broken()` for further details.

**get\_broken()** → bool

**init\_plot**(*p\_figure*: *Figure* = None, *p\_plot\_settings*: [PlotSettings](#) = None, *\*\*p\_kwargs*)

Initializes the plot functionalities of the class.

**Parameters**

- **p\_figure** (*Matplotlib.figure.Figure*, optional) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p\_plot\_settings** ([PlotSettings](#)) – Optional plot settings. If None, the default view is plotted (see attribute `C_PLOT_DEFAULT_VIEW`).

**update\_plot**(*\*\*p\_kwargs*)

Updates the plot.

**Parameters**

**\*\*p\_kwargs** – Implementation-specific plot data and/or parameters.

```
class mlpro.bf.systems.basics.MultiSystem(p_name: str = None, p_id=None, p_range_max=0,
                                         p_autorun=0, p_class_shared=<class
                                         'mlpro.bf.systems.basics.SystemShared'>, p_mode=0,
                                         p_latency: ~datetime.timedelta = None, p_t_step:
                                         ~datetime.timedelta = None, p_fct_strans:
                                         ~mlpro.bf.systems.basics.FctSTrans = None, p_fct_success:
                                         ~mlpro.bf.systems.basics.FctSuccess = None, p_fct_broken:
                                         ~mlpro.bf.systems.basics.FctBroken = None,
                                         p_mujoco_file=None, p_frame_skip: int = 1,
                                         p_state_mapping=None, p_action_mapping=None,
                                         p_camera_conf: tuple = (None, None, None), p_visualize:
                                         bool = False, p_logging=True, **p_kwargs)
```

Bases: [Workflow](#), [System](#)

A complex system of systems.

**Parameters**

- **p\_name**
- **p\_id**
- **p\_range\_max**



- `p_autorun`
- `p_class_shared`
- `p_mode`
- `p_latency`
- `p_t_step`
- `p_fct_strans`
- `p_fct_success`
- `p_fct_broken`
- `p_mujoco_file`
- `p_frame_skip`
- `p_state_mapping`
- `p_action_mapping`
- `p_camera_conf`
- `p_visualize`
- `p_logging`
- `p_kwargs`

`C_TYPE = 'Multi-System'`

`add_system(p_system: System, p_mappings)`

Adds sub system to the MultiSystem.

#### Parameters

- `p_system` (*System*) – The system to be added.
- `p_mappings` (*list*) – The mappings corresponding the system in the form : [ ((ip dim\_type, op dim\_type), (input\_sys\_id, input\_dim\_id), (op\_sys\_id, op\_dimension\_id)), ... ]

`_reset(p_seed=None) → None`

Resets the MultiSystem and all the sub-systems inside.

#### Parameters

- `p_seed` (*int*) – Seed for the purpose of reproducibility

`get_subsystem_ids()`

`get_subsystems()`

`get_subsystem(p_system_id) → System`

Returns a sub system from the MultiSystem.

#### Parameters

- `p_system_id` – Id of the system to be returned

#### Returns

- `sub_system` – The system to be returned by ID.

#### Return type

*System*

**get\_states()**

Returns a list of the states of all the Sub-Systems in the MultiSystem.

**Returns**

**states** – States of all the subsystems.

**Return type**

dict

**simulate\_reaction**(*p\_state*: [State](#) = None, *p\_action*: [Action](#) = None, *p\_t\_step*: *timedelta* = None) → [State](#)

Simulates the multisystem, based on the action and time step.

**Parameters**

- **p\_state** ([State](#)) – State of the system.
- **p\_action** ([Action](#).) – Action provided externally for the simulation of the system.

**Returns**

**current\_state** – The new state of the system after simulation.

**Return type**

[State](#)

**compute\_broken**(*p\_state*: [State](#)) → bool

Returns true if the system is broken

**compute\_success**(*p\_state*: [State](#)) → bool

Returns true if the system has reached success.

**Parameters**

**p\_state**

**Returns**

# TODO

**Return type**

Shall return true if any of the system returns true?

```
class mlpro.bf.systems.basics.DemoScenario(p_system: System, p_mode, p_action_pattern: str =
    'random', p_action: list = None, p_id=None,
    p_cycle_limit=0, p_auto_setup: bool = True, p_visualize:
    bool = True, p_logging=True)
```

Bases: [ScenarioBase](#)

Demo Scenario Class to demonstrate systems, inherits from the ScenarioBase.

**Parameters**

- **p\_system** ([System](#)) – Mandatory parameter, takes the system for the scenario.
- **p\_mode** – Operation mode. See Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM.
- **p\_action\_pattern** (str) – The action pattern to be used for demonstration. Default is C\_ACTION\_RANDOM
- **p\_action\_random** (list) – The action to be executed in constant mode. Mandatory when the mode is constant.
- **p\_id** – Optional external id
- **p\_cycle\_limit** (int) – Maximum number of cycles. Default = 0 (no limit).

- **p\_auto\_setup** (*bool*) – If True custom method setup() is called after initialization.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = True.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL.

**C\_NAME** = 'Demo System Scenario'

**C\_ACTION\_RANDOM** = 'random'

**C\_ACTION\_CONSTANT** = 'constant'

**setup()**

Set's up the system spaces.

**get\_latency()** → *timedelta*

Returns the latency of the system.

**\_reset**(*p\_seed*)

Resets the Scenario and the system. Sets up the action and state spaces of the system.

**Parameters**

**p\_seed** – Seed for the purpose of reproducibility.

**\_run\_cycle()**

Runs the custom scenario cycle, through the run method of the scenario base. Checks and returns the brokent state, false otherwise.

**\_get\_next\_action()**

Generates new action based on the pattern provided by the user.

**update\_plot**(\*\**p\_kwargs*)

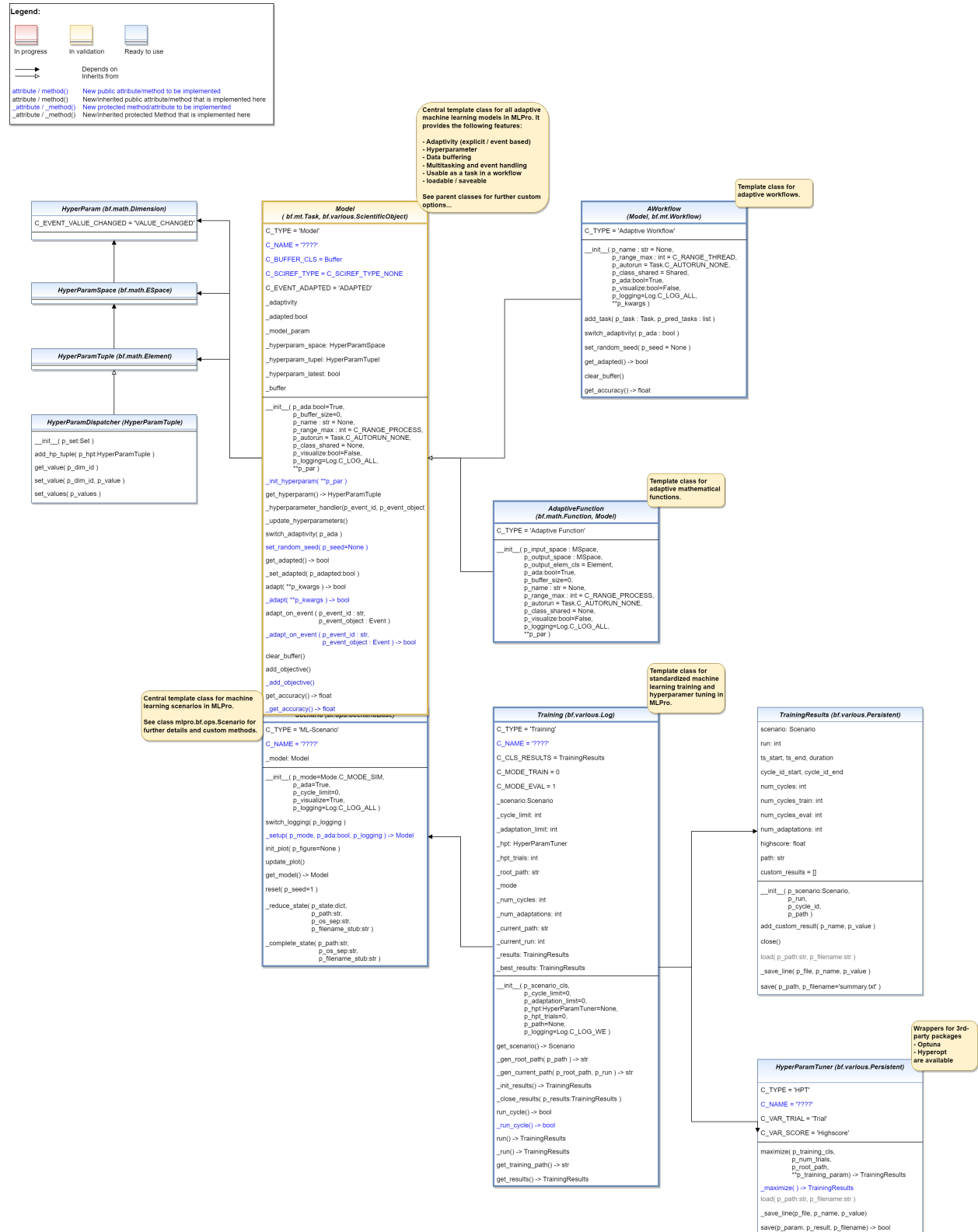
Updates the plot.

**Parameters**

**\*\*p\_kwargs** – Implementation-specific plot data and/or parameters.

## Layer 4 - Machine Learning

## BF-ML - Machine Learning



Ver. 2.3.0 (2023-07-24)

This module provides the fundamental templates and processes for machine learning in MLPro.

```
class mlpro.bf.ml.basics.HyperParam(p_name_short, p_base_set='R', p_name_long="", p_name_latex="",
                                     p_unit="", p_unit_latex="", p_boundaries: list = [], p_description="",
                                     p_symmetrical: bool = False, p_logging=False, **p_kwargs)
```

Bases: [Dimension](#)

Hyperparameter definition class. See class [Dimension](#) for further descriptions.

```
C_EVENT_VALUE_CHANGED = 'VALUE_CHANGED'
```

```
class mlpro.bf.ml.basics.HyperParamSpace
```

Bases: [ESpace](#)

Hyperparameter space, which is just an Euclidian space.

```
class mlpro.bf.ml.basics.HyperParamTuple(p_set: Set)
```

Bases: [Element](#)

Tuple of hyperparameters, which is an element of a hyperparameter space

```
set_value(p_dim_id, p_value)
```

```
class mlpro.bf.ml.basics.HyperParamDispatcher(p_set: Set)
```

Bases: [HyperParamTuple](#)

To dispatch multiple hp tuples into one tuple

```
add_hp_tuple(p_hpt: HyperParamTuple)
```

```
get_value(p_dim_id)
```

```
set_value(p_dim_id, p_value)
```

```
get_values()
```

```
set_values(p_values)
```

Overwrites the values of all components of the element.

#### Parameters

**dimensions.** (*p\_values* Something iterable with same length as number of element)

```
class mlpro.bf.ml.basics.Model(p_ada: bool = True, p_buffer_size: int = 0, p_id=None, p_name: str =
                               None, p_range_max: int = 2, p_autorun=0, p_class_shared=None,
                               p_visualize: bool = False, p_logging=True, **p_par)
```

Bases: [Task](#), [ScientificObject](#)

**Fundamental template class for adaptive ML models. Supports in particular**

- Adaptivity (explicit and/or event based)
- Hyperparameter management
- Data buffering
- Multitasking
- Plotting
- Scientific referencing on source code level

#### Parameters

- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_buffer\_size** (*int*) – Initial size of internal data buffer. Default = 0 (no buffering).
- **p\_id** – Optional external id
- **p\_name** (*str*) – Optional name of the model. Default is None.
- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class Range. Default is Range.C\_RANGE\_PROCESS.
- **p\_autorun** (*int*) – On value C\_AUTORUN\_RUN method run() is called immediately during instantiation. On value C\_AUTORUN\_LOOP method run\_loop() is called. Value C\_AUTORUN\_NONE (default) causes an object instantiation without starting further actions.
- **p\_class\_shared** – Optional class for a shared object (class Shared or a child class of it)
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL
- **p\_par** (*Dict*) – Further model specific hyperparameters (to be defined in child class).

**C\_TYPE** = 'Model'

**C\_NAME** = '????'

**C\_EVENT\_ADAPTED** = 'ADAPTED'

**C\_EVENT\_OBJECTIVE\_REACHED** = False

**C\_BUFFER\_CLS**

alias of Buffer

**C\_SCIREF\_TYPE** = None

**\_init\_hyperparam**(\*\**p\_par*)

Implementation specific hyperparameters can be added here. Please follow these steps: a) Add each hyperparameter as an object of type HyperParam to the internal hyperparameter

space object self.\_hyperparam\_space

b) Create hyperparameter tuple and bind to self.\_hyperparam\_tuple

c) Set default value for each hyperparameter

#### Parameters

**p\_par** (*Dict*) – Further model specific hyperparameters, that are passed through constructor.

**get\_hyperparam**() → *HyperParamTuple*

Returns the internal hyperparameter tuple to get access to single values.

**\_hyperparam\_handler**(*p\_event\_id*, *p\_event\_object*: *Event*)

This is an event handler method for managing the updates of hyperparameters in the HPTuple. Set's the flag hp\_latest to false, as the hpt of the model are not latest to the tuple.

**\_update\_hyperparameters**() → bool

Custom method to update the hyperparameters of a system with the latest value in the Hyperparameter Tuple. This may be due to Hyperparameter Tuning. Please return True if the hyperparameters are updated successfully.

**Returns**

True if the hyperparameters are updated successfully.

**Return type**

bool

**switch\_adaptivity**(*p\_ada: bool*)

Switches adaption functionality on/off.

**Parameters**

**p\_ada** (*bool*) – Boolean switch for adaptivity

**set\_random\_seed**(*p\_seed=None*)

Resets the internal random generator using the given seed.

**get\_adapted**() → bool

Returns True, if the model was adapted at least once. False otherwise.

**\_set\_adapted**(*p\_adapted: bool*)

Sets the adapted flag. Internal use only.

**adapt**(\*\**p\_kwargs*) → bool

Adapts the model by calling the custom method `_adapt()`.

**Parameters**

**p\_kwargs** (*dict*) – All parameters that are needed for the adaption. Depends on the specific higher context.

**Returns**

**adapted** – True, if something has been adapted. False otherwise.

**Return type**

bool

**\_adapt**(\*\**p\_kwargs*) → bool

Custom implementation of the adaptation algorithm. Please specify the parameters needed by your implementation. This method will be called by public method `adapt()` if adaptivity is switched on.

**Parameters**

**p\_kwargs** (*dict*) – All parameters that are needed for the adaption. Please replace by concrete parameter definitions that meet the needs of your algorithm.

**Returns**

**adapted** – True, if something has been adapted. False otherwise.

**Return type**

bool

**adapt\_on\_event**(*p\_event\_id: str, p\_event\_object: Event*)

Method to be used as event handler for event-based adaptations. Calls custom method `_adapt_on_event()` and updates the internal adaptation state.

**Parameters**

- **p\_event\_id** (*str*) – Event id.
- **p\_event\_object** (*Event*) – Object with further context informations about the event.

**\_adapt\_on\_event**(*p\_event\_id: str, p\_event\_object: Event*) → bool

Custom method to be used for event-based adaptation. See method `adapt_on_event()`.

**Parameters**



- **p\_event\_id** (*str*) – Event id.
- **p\_event\_object** ([Event](#)) – Object with further context informations about the event.

**Returns**

**adapted** – True, if something was adapted. False otherwise.

**Return type**

bool

**clear\_buffer()**

Clears internal buffer (if buffering is active).

**add\_objective(\*\*p\_kwargs)**

Determines the objective of the model.

**\_add\_objective(\*\*p\_kwargs)**

This method is called in `add_objective()`. Please redefine this method.

**get\_accuracy() → float**

Determines the accuracy of the model.

**Returns**

**accuracy** – Accuracy of the model as a scalar value in interval [0,1]

**Return type**

float

**\_get\_accuracy() → float**

This method is called in `get_accuracy()`. Please redefine this method.

**Returns**

**accuracy** – Accuracy of the model as a scalar value in interval [0,1]

**Return type**

float

```
class mlpro.bf.ml.basics.AWorkflow(p_name: str = None, p_range_max=1, p_class_shared=<class
    'mlpro.bf.ml.Shared'>, p_ada: bool = True, p_visualize: bool = False,
    p_logging=True, **p_kwargs)
```

Bases: [Model](#), [Workflow](#)

Adaptive workflow based on a workflow and an adaptive ml model.

**Parameters**

- **p\_name** (*str*) – Optional name of the workflow. Default is None.
- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class `Range`. Default is `Range.C_RANGE_PROCESS`.
- **p\_class\_shared** – Optional class for a shared object (class `OAShared` or a child class of `OAShared`)
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class `Log`). Default: `Log.C_LOG_ALL`
- **p\_kwargs** (*dict*) – Further optional named parameters.

**C\_TYPE** = 'Adaptive Workflow'

**add\_task**(*p\_task*: [Task](#), *p\_pred\_tasks*: *list = None*)

Adds a task to the workflow.

**Parameters**

- **p\_task** ([Task](#)) – Task object to be added.
- **p\_pred\_tasks** (*list*) – Optional list of predecessor task objects

**switch\_adaptivity**(*p\_ada*: *bool*)

Switches adaption functionality on/off.

**Parameters**

**p\_ada** (*bool*) – Boolean switch for adaptivity

**set\_random\_seed**(*p\_seed*=*None*)

Resets the internal random generator using the given seed.

**get\_adapted**() → *bool*

Returns True, if the model was adapted at least once. False otherwise.

**clear\_buffer**()

Clears internal buffer (if buffering is active).

**get\_accuracy**()

Determines the accuracy of the model.

**Returns**

**accuracy** – Accuracy of the model as a scalar value in interval [0,1]

**Return type**

float

**class** mlpro.bf.ml.basics.**Scenario**(*p\_mode*=0, *p\_ada*: *bool = True*, *p\_cycle\_limit*: *int = 0*, *p\_auto\_setup*:  
*bool = True*, *p\_visualize*: *bool = True*, *p\_logging*=*True*)

Bases: [ScenarioBase](#)

Template class for a common ML scenario with an adaptive model inside. To be inherited and specialized in higher ML subtopic layers. See class bf.ops.ScenarioBase for further details and custom methods.

**The following key features are included:**

- Operation mode
- Cycle management
- Timer
- Latency
- Explicit handling of an adaptive ML model inside

**Parameters**

- **p\_mode** – Operation mode. See bf.ops.Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM.
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_cycle\_limit** (*int*) – Maximum number of cycles. Default = 0 (no limit).
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = True.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL.

**C\_TYPE** = 'ML-Scenario'

**C\_NAME** = '????'

**switch\_logging**(*p\_logging*)

Sets new log level.

**Parameters**

**p\_logging** – Log level (constant C\_LOG\_LEVELS contains valid values)

**setup**()

Custom method to set up all components of the scenario.

**\_setup**(*p\_mode*, *p\_ada*: bool, *p\_visualize*: bool, *p\_logging*) → *Model*

Custom setup of ML scenario.

**Parameters**

- **p\_mode** – Operation mode. See Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM
- **p\_ada** (bool) – Boolean switch for adaptivity.
- **p\_visualize** (bool) – Boolean switch for visualisation.
- **p\_logging** – Log level (see constants of class Log).

**Returns**

**model** – Adaptive model inside the ML scenario

**Return type**

*Model*

**init\_plot**(*p\_figure*: Figure = None, *p\_plot\_settings*: PlotSettings = None)

Initializes the plot functionalities of the class.

**Parameters**

- **p\_figure** (*Matplotlib.figure.Figure*, optional) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p\_plot\_settings** (*PlotSettings*) – Optional plot settings. If None, the default view is plotted (see attribute C\_PLOT\_DEFAULT\_VIEW).

**update\_plot**(\*\**p\_kwargs*)

Updates the plot.

**Parameters**

**\*\*p\_kwargs** – Implementation-specific plot data and/or parameters.

**get\_model**() → *Model*

Returns the adaptive model object inside the scenario.

**reset**(*p\_seed*=1)

Resets the scenario and especially the ML model inside. Internal random generators are seed with the given value. Custom reset actions can be implemented in method \_reset().

**Parameters**

**p\_seed** (int) – Seed value for internal random generator

**`_reduce_state(p_state: dict, p_path: str, p_os_sep: str, p_filename_stub: str)`**

Custom method to reduce the given object state by components that can not be pickled. Further data files can be created in the given path and should use the given filename stub.

**Parameters**

- **`p_state`** (*dict*) – Object state dictionary to be reduced by components that can not be pickled.
- **`p_path`** (*str*) – Path to store further optional custom data files
- **`p_os_sep`** (*str*) – OS-specific path separator.
- **`p_filename_stub`** (*str*) – Filename stub to be used for further optional custom data files

**`_complete_state(p_path: str, p_os_sep: str, p_filename_stub: str)`**

Custom method to complete the object state (=self) from external data sources. This method is called by standard method `__setstate__()` during unpickling the object from an external file.

**Parameters**

- **`p_path`** (*str*) – Path of the object pickle file (and further optional related files)
- **`p_os_sep`** (*str*) – OS-specific path separator.
- **`p_filename_stub`** (*str*) – Filename stub to be used for further optional custom data files

**`class mlpro.bf.ml.basics.TrainingResults(p_scenario: Scenario, p_run, p_cycle_id, p_logging='W')`**

Bases: [Persistent](#)

Results of a training (see class [Training](#)).

**Parameters**

- **`p_scenario`** ([Scenario](#)) – Related scenario.
- **`p_run`** (*int*) – Run id.
- **`p_cycle_id`** (*int*) – Id of first cycle of this run.
- **`p_logging`** – Log level (see constants of class [Log](#)). Default: `Log.C_LOG_ALL`

**`C_TYPE = 'Results '`**

**`add_custom_result(p_name, p_value)`**

**`close()`**

**`log_results()`**

**`_log_results()`**

**`classmethod load(p_path: str, p_filename: str)`**

Loading training results is explicitly disabled.

**`_save_line(p_file, p_name, p_value)`**

**`save(p_path: str, p_filename: str = 'summary.csv') → bool`**

Saves a training summary in the given path.

**Parameters**

- **`p_path`** (*str*) – Destination folder
- **`p_filename`** (*string*) – Name of summary file. Default = 'summary.csv'

**Returns**

**success** – True, if summary file was created successfully. False otherwise.

**Return type**

bool

**class** mlpro.bf.ml.basics.**HyperParamTuner**(*p\_id=None, p\_logging=True*)

Bases: *Persistent*

Template class for hyperparameter tuning (HPT).

**C\_TYPE** = 'HyperParam Tuner'

**C\_NAME** = '????'

**C\_VAR\_TRIAL** = 'Trial'

**C\_VAR\_SCORE** = 'Highscore'

**maximize**(*p\_training\_cls, p\_num\_trials, p\_root\_path, \*\*p\_training\_param*) → *TrainingResults*

...

**Parameters**

- **p\_training\_cls** – Training class to be instantiated/executed
- **p\_num\_trials** (*str*) – Number of trials
- **p\_root\_path** – Root path of the training class
- **p\_training\_param** (*dictionary*) – Training parameters

**Returns**

**results** – Training results of the best tuned model (see class *TrainingResults*).

**Return type**

*TrainingResults*

**\_maximize**() → *TrainingResults*

**classmethod load**(*p\_path: str, p\_filename: str*)

Loading training results is explicitly disabled.

**\_save\_line**(*p\_file, p\_name, p\_value*)

**save**(*p\_param, p\_result, p\_filename='best\_parameters.csv'*) → bool

Saves the best result of the hyperparameter tuning in the root path.

**Parameters**

- **p\_param** (*dict*) – A dictionary that consists of list of best parameters
- **p\_result** (*float*) – Highest score
- **p\_filename** (*str*) – Name of summary file. Default = 'best\_parameters.csv'

**Returns**

**success** – True, if summary file was created successfully. False otherwise.

**Return type**

bool

```
class mlpro.bf.ml.basics.Training(**p_kwargs)
```

Bases: [Log](#)

Template class for a ML training and hyperparameter tuning.

#### Parameters

- **p\_scenario\_cls** – Name of ML scenario class, compatible to/inherited from class Scenario.
- **p\_cycle\_limit** (*int*) – Maximum number of training cycles (0=no limit). Default = 0.
- **p\_adaptation\_limit** (*int*) – Maximum number of adaptations (0=no limit). Default = 0.
- **p\_hpt** ([HyperParamTuner](#)) – Optional hyperparameter tuner (see class HyperParamTuner). Default = None.
- **p\_hpt\_trials** (*int*) – Optional number of hyperparameter tuning trials. Default = 0.
- **p\_path** (*str*) – Optional destination path to store training data. Default = None.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default = Log.C\_LOG\_WE.

```
C_TYPE = 'Training'
```

```
C_NAME = '????'
```

```
C_CLS_RESULTS
```

alias of [TrainingResults](#)

```
C_MODE_TRAIN = 0
```

```
C_MODE_EVAL = 1
```

```
C_LOG_SEPARATOR =
```

```
'-----'
```

```
get_scenario() → Scenario
```

```
_gen_root_path(p_path) → str
```

```
_gen_current_path(p_root_path, p_run) → str
```

```
_init_results() → TrainingResults
```

```
_close_results(p_results: TrainingResults)
```

```
run_cycle() → bool
```

Runs a single training cycle.

#### Returns

**termination\_event** – True, if training run has finished. False otherwise.

#### Return type

bool

```
_run_cycle() → bool
```

Single custom training cycle to be redefined. Custom training results can be added to using `self._results.add_custom_result(p_name, p_value)`.

#### Returns

True, if training has finished. False otherwise.

**Return type**

bool

**run()** → *TrainingResults*

Runs a training and returns the results of the best trained/tuned agent.

**Returns**

Object with training results.

**Return type***TrainingResults***\_run()** → *TrainingResults***get\_results()** → *TrainingResults***get\_training\_path()** → str

```
class mlpro.bf.ml.basics.AdaptiveFunction(p_input_space: ~mlpro.bf.math.basics.MSpace,
                                          p_output_space: ~mlpro.bf.math.basics.MSpace,
                                          p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>,
                                          p_ada: bool = True, p_buffer_size: int = 0, p_name: str =
                                          None, p_range_max: int = 2, p_autorun=0,
                                          p_class_shared=None, p_visualize: bool = False,
                                          p_logging=True, **p_par)
```

Bases: *Function*, *Model*

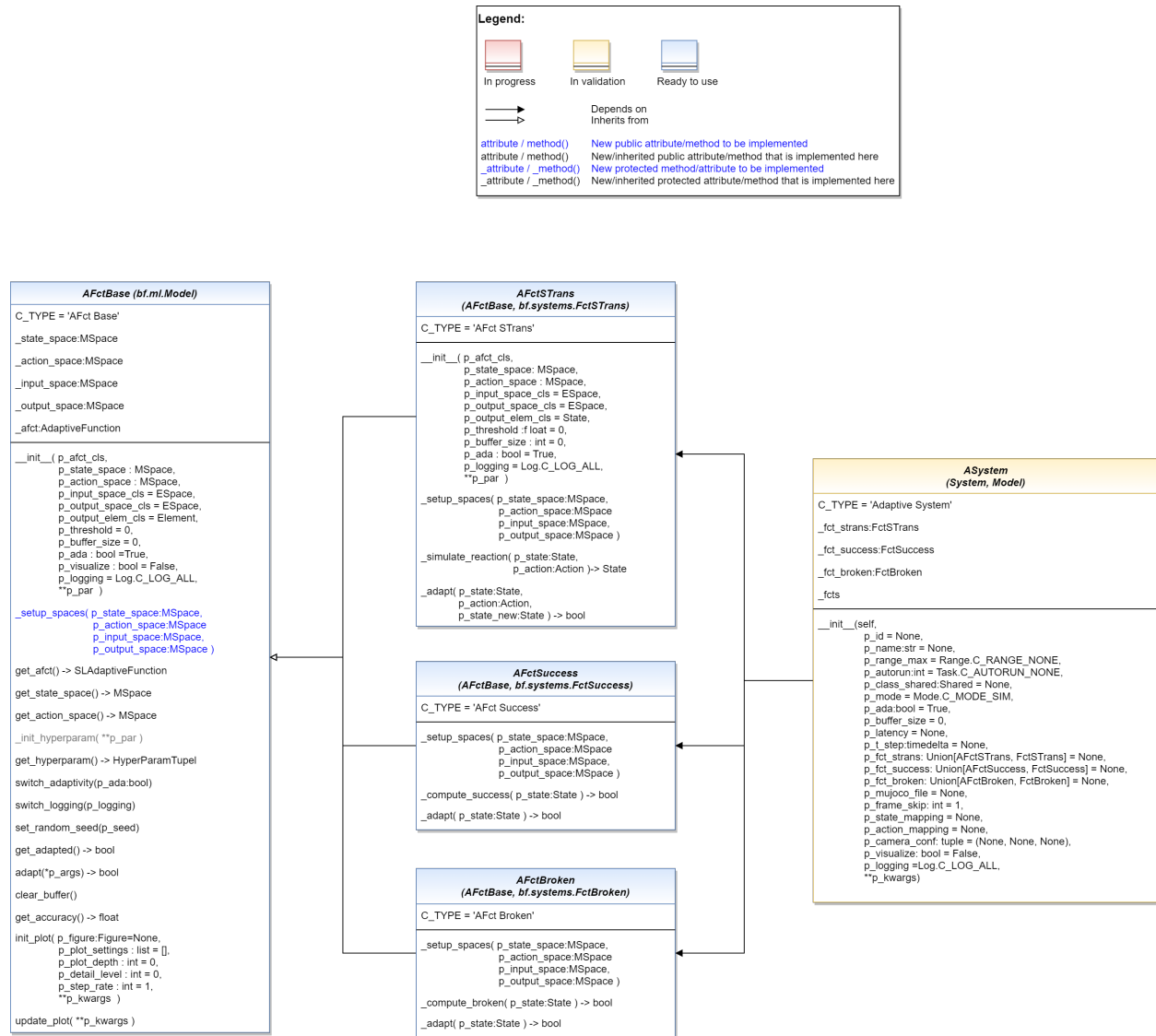
Template class for an adaptive bi-multivariate mathematical function. The kind of adaptation (learning paradigm) is to be specified in child classes.

**Parameters**

- **p\_input\_space** (*MSpace*) – Input space of function
- **p\_output\_space** (*MSpace*) – Output space of function
- **p\_output\_elem\_cls** – Output element class (compatible to/inherited from class *Element*)
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_buffer\_size** (*int*) – Initial size of internal data buffer. Default = 0 (no buffering).
- **p\_name** (*str*) – Optional name of the model. Default is None.
- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class *Range*. Default is *Range.C\_RANGE\_PROCESS*.
- **p\_autorun** (*int*) – On value *C\_AUTORUN\_RUN* method *run()* is called immediately during instantiation. On value *C\_AUTORUN\_LOOP* method *run\_loop()* is called. Value *C\_AUTORUN\_NONE* (default) causes an object instantiation without starting further actions.
- **p\_class\_shared** – Optional class for a shared object (class *Shared* or a child class of it)
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class *Log*). Default: *Log.C\_LOG\_ALL*
- **p\_par** (*Dict*) – Further model specific hyperparameters (to be defined in child class).

**C\_TYPE** = 'Adaptive Function'**C\_NAME** = '????'

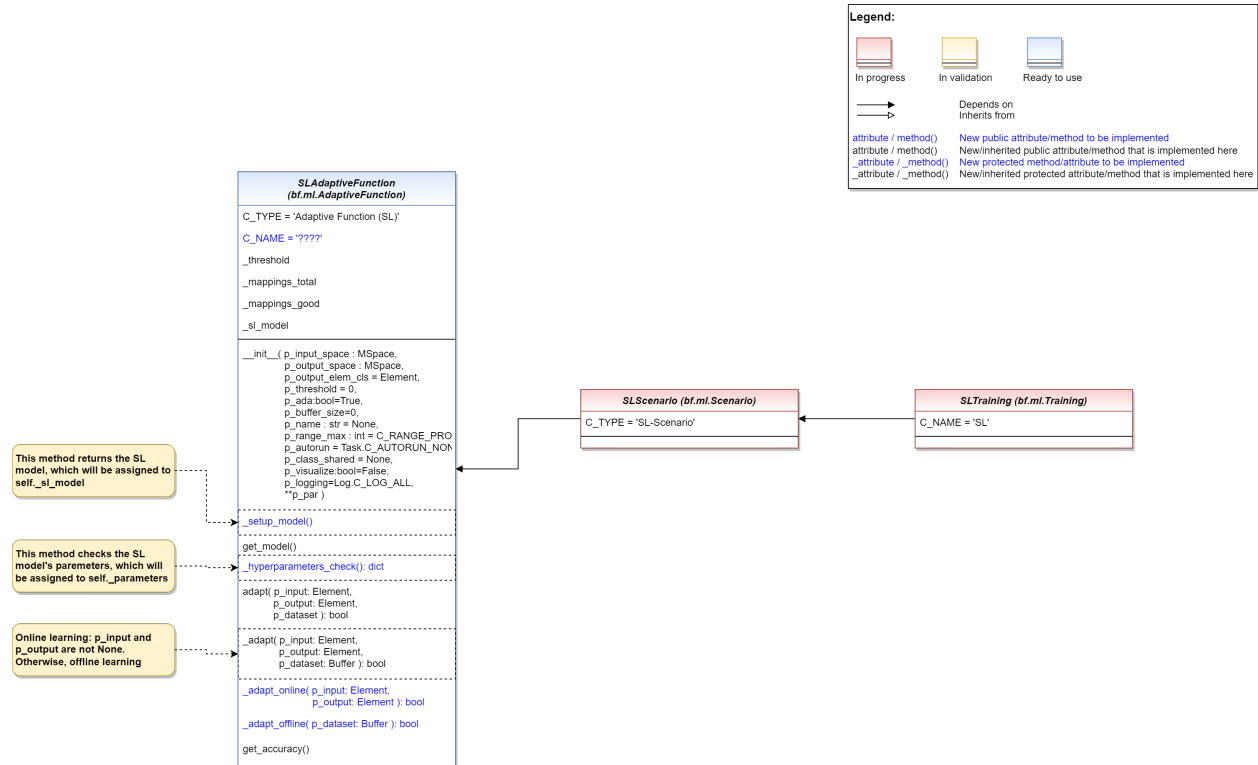
## BF-ML-SYSTEMS - Adaptive State-based Systems



## 11.1.2 MLPro-SL - Supervised Learning



## SL - Basics



Ver. 0.5.1 (2023-06-24)

This module provides model classes for supervised learning tasks.

```

class mlpro.sl.basics.SLAdaptiveFunction(p_input_space: ~mlpro.bf.math.basics.MSpace,
                                         p_output_space: ~mlpro.bf.math.basics.MSpace,
                                         p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>,
                                         p_threshold=0, p_ada: bool = True, p_buffer_size: int = 0,
                                         p_metrics: ~typing.List[~mlpro.sl.models_eval.Metric] = [],
                                         p_score_metric=None, p_name: str = None, p_range_max: int
                                         = 2, p_autorun=0, p_class_shared=None, p_visualize: bool =
                                         False, p_logging=True, **p_par)
  
```

Bases: [AdaptiveFunction](#)

Template class for an adaptive bi-multivariate mathematical function that adapts by supervised learning.

**Parameters**

- **p\_input\_space** ([MSpace](#)) – Input space of function
- **p\_output\_space** ([MSpace](#)) – Output space of function
- **p\_output\_elem\_cls** – Output element class (compatible to/inherited from class [Element](#))
- **p\_threshold** (*float*) – Threshold for the difference between a setpoint and a computed output. Computed outputs with a difference less than this threshold will be assessed as 'good' outputs. Default = 0.
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_buffer\_size** (*int*) – Initial size of internal data buffer. Default = 0 (no buffering).
- **p\_name** (*str*) – Optional name of the model. Default is None.

- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class Range. Default is Range.C\_RANGE\_PROCESS.
- **p\_autorun** (*int*) – On value C\_AUTORUN\_RUN method run() is called immediately during instantiation. On value C\_AUTORUN\_LOOP method run\_loop() is called. Value C\_AUTORUN\_NONE (default) causes an object instantiation without starting further actions.
- **p\_class\_shared** – Optional class for a shared object (class Shared or a child class of it)
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL
- **p\_par** (*Dict*) – Further model specific hyperparameters (to be defined in child class).

**C\_TYPE** = 'Adaptive Function (SL)'

**C\_NAME** = '????'

**\_setup\_model()**

A method to set up a supervised learning network. Please redefine this method according to the type of network, if not provided yet.

**Return type**

A set up supervised learning model

**get\_input\_space()**

**get\_output\_space()**

**\_setup\_logging\_set()** → *Set*

**get\_logging\_set()** → *Set*

**get\_logging\_data()** → *list*

**get\_model()**

A method to get the supervised learning network.

**Return type**

A set up supervised learning model

**adapt**(*p\_input*: *Element* = None, *p\_output*: *Element* = None, *p\_dataset*=None) → *bool*

Adaption by supervised learning.

**Parameters**

- **p\_input** (*Element*) – Abscissa/input element object (type Element)
- **p\_output** (*Element*) – Setpoint ordinate/output element (type Element)
- **p\_dataset** – A set of data for offline learning

**\_adapt**(*p\_input*: *Element*, *p\_output*: *Element*, *p\_dataset*: *Buffer*) → *bool*

Adaptation algorithm that is called by public adaptation method. This covers online and offline supervised learning. Online learning means that the data set is dynamic according to the input data, meanwhile offline learning means that the learning procedure is based on a static data set.

**Parameters**

- **p\_input** (*Element*) – Abscissa/input element object (type Element) for online learning

- **p\_output** (*Element*) – Setpoint ordinate/output element (type *Element*) for online learning
- **p\_dataset** (*Buffer*) – A set of data for offline learning

**Return type**  
bool

**\_adapt\_online**(*p\_input*: *Element*, *p\_output*: *Element*) → bool

Custom adaptation algorithm for online learning. Please redefine.

**Parameters**

- **p\_input** (*Element*) – Abscissa/input element object (type *Element*)
- **p\_output** (*Element*) – Setpoint ordinate/output element (type *Element*)

**Return type**  
bool

**\_adapt\_offline**(*p\_dataset*: *Buffer*) → bool

Custom adaptation algorithm for offline learning. Please redefine.

**Parameters**

**p\_dataset** (*Buffer*) – A set of data for offline learning

**Return type**  
bool

**get\_accuracy()**

Returns the accuracy of the adaptive function. The accuracy is defined as the relation between the number of successful mapped inputs and the total number of mappings since the last adaptation.

**get\_metrics()** → list

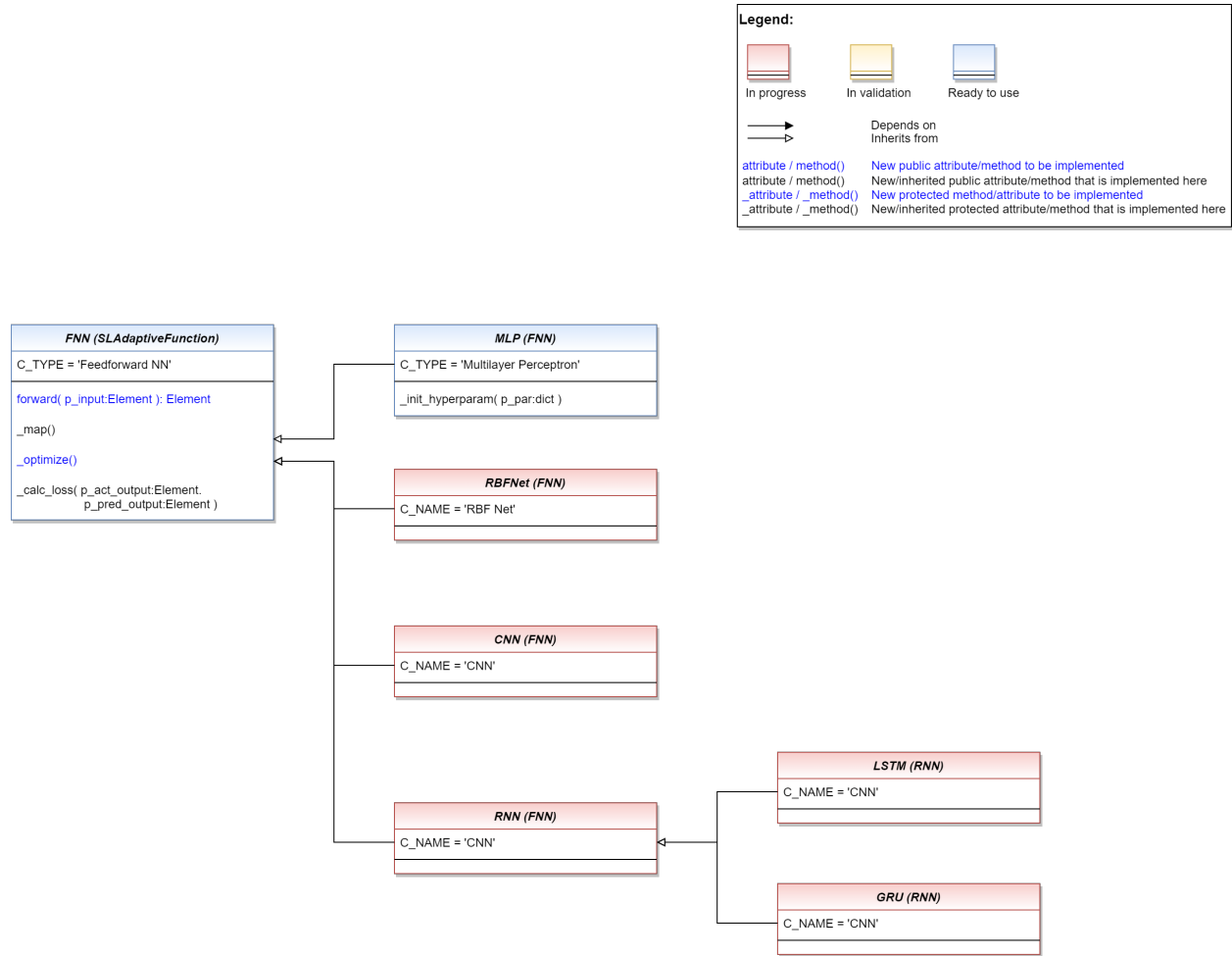
**get\_score\_metric()**

**get\_metric\_space()** → *ESpace*

**calculate\_metrics**(*p\_data*) → *Element*

**get\_current\_metrics()**

## SL - Feedforward Neural Network



Ver. 1.1.0 (2023-03-10)

This module provides model classes of feedforward neural networks for supervised learning tasks.

```

class mlpro.sl.fnn.FNN(p_input_space: ~mlpro.bf.math.basics.MSpace, p_output_space:
    ~mlpro.bf.math.basics.MSpace, p_output_elem_cls=<class
    'mlpro.bf.math.basics.Element'>, p_threshold=0, p_ada: bool = True, p_buffer_size:
    int = 0, p_metrics: ~typing.List[~mlpro.sl.models_eval.Metric] = [],
    p_score_metric=None, p_name: str = None, p_range_max: int = 2, p_autorun=0,
    p_class_shared=None, p_visualize: bool = False, p_logging=True, **p_par)
  
```

Bases: [SLAdaptiveFunction](#)

This class provides the base class of feedforward neural networks.

**C\_TYPE = 'Feedforward NN'**

**forward**(p\_input: [Element](#)) → [Element](#)

Custom forward propagation in neural networks to generate some output that can be called by an external method. Please redefine.

**Parameters**

**p\_input** ([Element](#)) – Input data

**Returns****output** – Output data**Return type***Element***\_map**(*p\_input*: *Element*, *p\_output*: *Element*)

Maps a multivariate abscissa/input element to a multivariate ordinate/output element.

**Parameters**

- **p\_input** (*Element*) – Abscissa/input element object (type *Element*)
- **p\_output** (*Element*) – Setpoint ordinate/output element (type *Element*)

**\_optimize**()

This method provides provide a funtionalitiy to call the optimizer of the feedforward network.

**\_calc\_loss**(*p\_act\_output*: *Element*, *p\_pred\_output*: *Element*)

This method provides provide a funtionalitiy to call the loss function of the feedforward network.

**Parameters**

- **p\_act\_output** (*Element*) – Actual output from the buffer.
- **p\_pred\_output** (*Element*) – Predicted output by the SL model.

```
class mlpro.sl.fnn.MLP(p_input_space: ~mlpro.bf.math.basics.MSpace, p_output_space:
    ~mlpro.bf.math.basics.MSpace, p_output_elem_cls=<class
    'mlpro.bf.math.basics.Element'>, p_threshold=0, p_ada: bool = True, p_buffer_size:
    int = 0, p_metrics: ~typing.List[~mlpro.sl.models_eval.Metric] = [],
    p_score_metric=None, p_name: str = None, p_range_max: int = 2, p_autorun=0,
    p_class_shared=None, p_visualize: bool = False, p_logging=True, **p_par)
```

Bases: *FNN*

This class provides the base class of multilayer perceptron.

**C\_TYPE** = 'Multilayer Perceptron'**\_init\_hyperparam**(\*\**p\_par*)

A method to deal with the hyperparameters related to the MLP model.

**Hyperparameters****p\_update\_rate** :

update rate.

**p\_num\_hidden\_layers** :

number of hidden layers.

**p\_hidden\_size** :

number of hidden neurons.

**p\_activation\_fct** :

activation function.

**p\_output\_activation\_fct** :

extra activation function for the output layer.

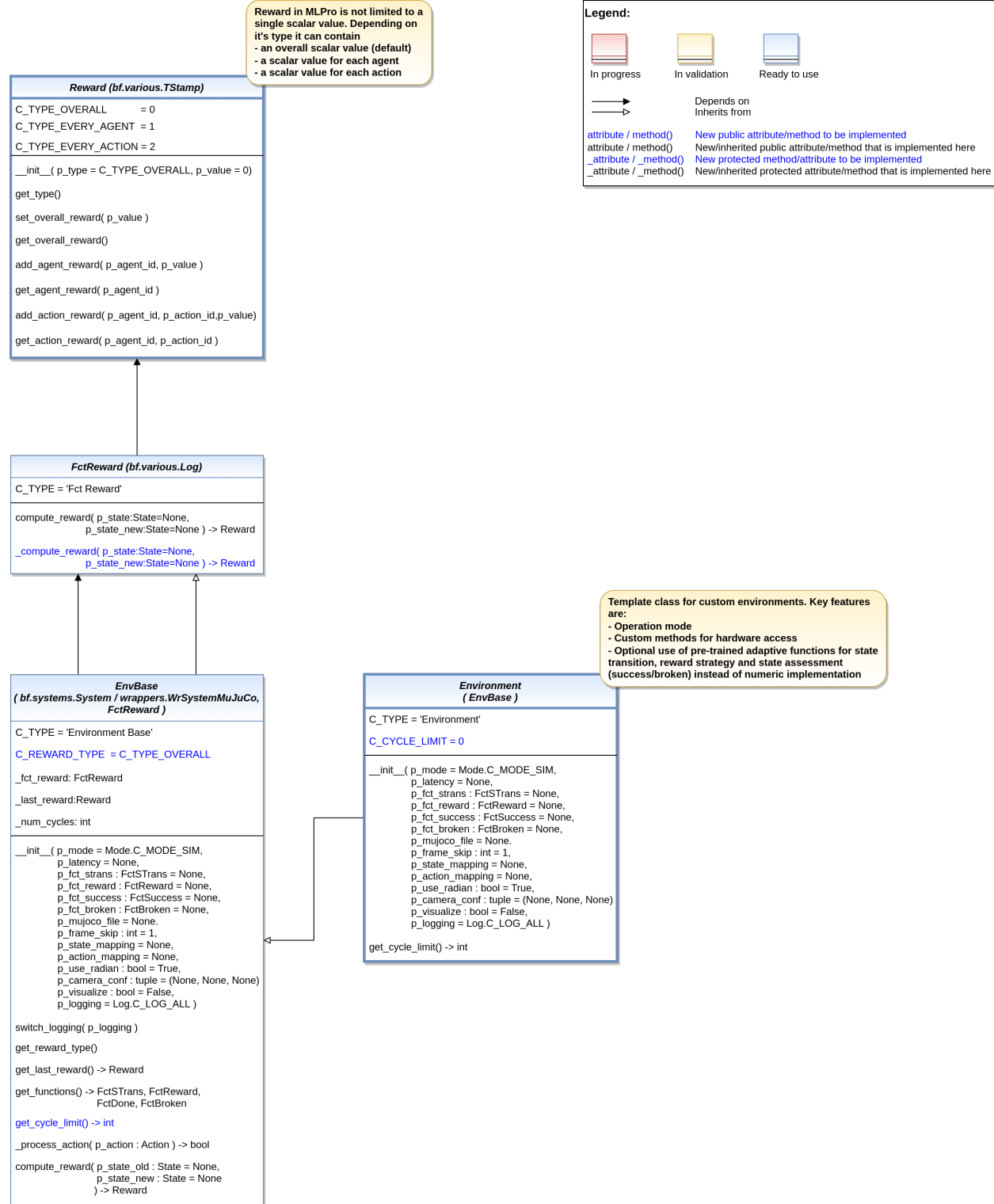
**p\_optimizer** :

optimizer.

**p\_loss\_fct :**  
loss function.

### 11.1.3 MLPro-RL - Reinforcement Learning

## RL-ENV - Environments



Ver. 1.7.4 (2023-05-30)

This module provides model classes for environments.

```
class mlpro.rl.models_env.Reward(p_type=0, p_value=0)
```

Bases: [TStamp](#)

Objects of this class represent rewards of environments. The internal structure depends on the reward type. Three types are supported as listed below.

#### Parameters

- **p\_type** – Reward type (default: C\_TYPE\_OVERALL)
- **p\_value** – Overall reward value (reward type C\_TYPE\_OVERALL only)

**C\_TYPE\_OVERALL** = 0

**C\_TYPE\_EVERY\_AGENT** = 1

**C\_TYPE\_EVERY\_ACTION** = 2

**C\_VALID\_TYPES** = [0, 1, 2]

**get\_type()**

**is\_rewarded**(p\_agent\_id) → bool

**set\_overall\_reward**(p\_reward) → bool

**get\_overall\_reward()**

**add\_agent\_reward**(p\_agent\_id, p\_reward) → bool

**get\_agent\_reward**(p\_agent\_id)

**add\_action\_reward**(p\_agent\_id, p\_action\_id, p\_reward) → bool

**get\_action\_reward**(p\_agent\_id, p\_action\_id)

**class** mlpro.rl.models\_env.FctReward(p\_logging=True)

Bases: [Log](#)

Template class for reward functions.

#### Parameters

- **p\_logging** – Log level (see class Log for more details).

**C\_TYPE** = 'Fct Reward'

**compute\_reward**(p\_state\_old: [State](#) = None, p\_state\_new: [State](#) = None) → [Reward](#)

Computes a reward based on a predecessor and successor state. Custom method `_compute_reward()` is called.

#### Parameters

- **p\_state\_old** ([State](#)) – Predecessor state.
- **p\_state\_new** ([State](#)) – Successor state.

#### Returns

**r** – Reward

#### Return type

[Reward](#)



**\_compute\_reward**(*p\_state\_old*: *State* = None, *p\_state\_new*: *State* = None) → *Reward*

Custom reward method. See method `compute_reward()` for further details.

```
class mlpro.rl.models_env.EnvBase(p_mode=0, p_latency: timedelta = None, p_fct_strans: FctSTrans =
    None, p_fct_reward: FctReward = None, p_fct_success: FctSuccess =
    None, p_fct_broken: FctBroken = None, p_mujoco_file=None,
    p_frame_skip: int = 1, p_state_mapping=None,
    p_action_mapping=None, p_camera_conf: tuple = (None, None, None),
    p_visualize: bool = False, p_logging=True)
```

Bases: *FctReward*, *System*

Base class for all environment classes. It defines the interface and elementary properties for an environment in the context of reinforcement learning.

#### Parameters

- **p\_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant `C_LATENCY` is used by default.
- **p\_fct\_strans** (*FctSTrans*) – Optional external function for state transition.
- **p\_fct\_reward** (*FctReward*) – Optional external function for reward computation.
- **p\_fct\_success** (*FctSuccess*) – Optional external function for state evaluation ‘success’.
- **p\_fct\_broken** (*FctBroken*) – Optional external function for state evaluation ‘broken’.
- **p\_mujoco\_file** – Path to XML file for MuJoCo model.
- **p\_frame\_skip** (*int*) – Frame to be skipped every step. Default = 1.
- **p\_state\_mapping** – State mapping if the MLPro state and MuJoCo state have different naming.
- **p\_action\_mapping** – Action mapping if the MLPro action and MuJoCo action have different naming.
- **p\_use\_radian** (*bool*) – Use radian if the action and the state based on radian unit. Default = True.
- **p\_camera\_conf** (*tuple*) – Default camera configuration on MuJoCo Simulation (xyz position, elevation, distance).
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see class `Log` for more details).

#### **\_latency**

Latency of the environment.

##### Type

*timedelta*

#### **\_state**

Current state of environment.

##### Type

*State*

#### **\_state**

Previous state of environment.

##### Type

*State*

**\_last\_action**

Last action.

**Type**

*Action*

**\_last\_reward**

Last reward.

**Type**

*Reward*

**\_afct\_strans**

Internal adaptive state transition function.

**Type**

AFctSTrans

**\_afct\_reward**

Internal adaptive reward function.

**Type**

*AFctReward*

**\_afct\_success**

Internal adaptive function for state evaluation 'success'.

**Type**

AFctSuccess

**\_afct\_broken**

Internal adaptive function for state evaluation 'broken'.

**Type**

AFctBroken

**C\_TYPE = 'Environment Base'**

**C\_REWARD\_TYPE = 0**

**switch\_logging(*p\_logging*)**

Sets new log level.

**Parameters**

**p\_logging** – Log level (constant C\_LOG\_LEVELS contains valid values)

**get\_reward\_type()**

**get\_last\_reward()** → *Reward*

**get\_functions()**

**get\_cycle\_limit()** → int

Returns limit of cycles per training episode. To be implemented in child classes.

**\_process\_action(*p\_action*: *Action*)** → bool

Custom method for state transition. To be implemented in a child class. See method process\_action() for further details.

**compute\_reward**(*p\_state\_old*: [State](#) = None, *p\_state\_new*: [State](#) = None) → [Reward](#)

Computes a reward for the state transition, given by two successive states. The reward computation itself is carried out either by a custom implementation in method `_compute_reward()` or by an embedded adaptive function.

#### Parameters

- **p\_state\_old** ([State](#)) – Optional state before transition. If None the internal previous state of the environment is used.
- **p\_state\_new** ([State](#)) – Optional state after transition. If None the internal current state of the environment is used.

#### Returns

Reward object.

#### Return type

[Reward](#)

```
class mlpro.rl.models_env.Environment(p_mode=0, p_latency: timedelta = None, p_fct_strans: FctSTrans
                                     = None, p_fct_reward: FctReward = None, p_fct_success:
                                     FctSuccess = None, p_fct_broken: FctBroken = None,
                                     p_mujoco_file=None, p_frame_skip: int = 1,
                                     p_state_mapping=None, p_action_mapping=None,
                                     p_camera_conf: tuple = (None, None, None), p_visualize: bool =
                                     False, p_logging=True)
```

Bases: [EnvBase](#)

This class represents the central environment model to be reused/inherited in own rl projects.

#### Parameters

- **p\_mode** – Mode of environment. Possible values are `Mode.C_MODE_SIM`(default) or `Mode.C_MODE_REAL`.
- **p\_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant `C_LATENCY` is used by default.
- **p\_fct\_strans** ([FctSTrans](#)) – Optional external function for state transition.
- **p\_fct\_reward** ([FctReward](#)) – Optional external function for reward computation.
- **p\_fct\_success** ([FctSuccess](#)) – Optional external function for state evaluation ‘success’.
- **p\_fct\_broken** ([FctBroken](#)) – Optional external function for state evaluation ‘broken’.
- **p\_mujoco\_file** – Path to XML file for MuJoCo model.
- **p\_frame\_skip** (*int*) – Frame to be skipped every step. Default = 1.
- **p\_state\_mapping** – State mapping if the MLPro state and MuJoCo state have different naming.
- **p\_action\_mapping** – Action mapping if the MLPro action and MuJoCo action have different naming.
- **p\_use\_radian** (*bool*) – Use radian if the action and the state based on radian unit. Default = True.
- **p\_camera\_conf** (*tuple*) – Default camera configuration on MuJoCo Simulation (xyz position, elevation, distance).
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = True.

- **p\_logging** – Log level (see class Log for more details)

**C\_TYPE** = 'Environment'

**C\_CYCLE\_LIMIT** = 0

**static setup\_spaces()**

Static template method to set up and return state and action space of environment.

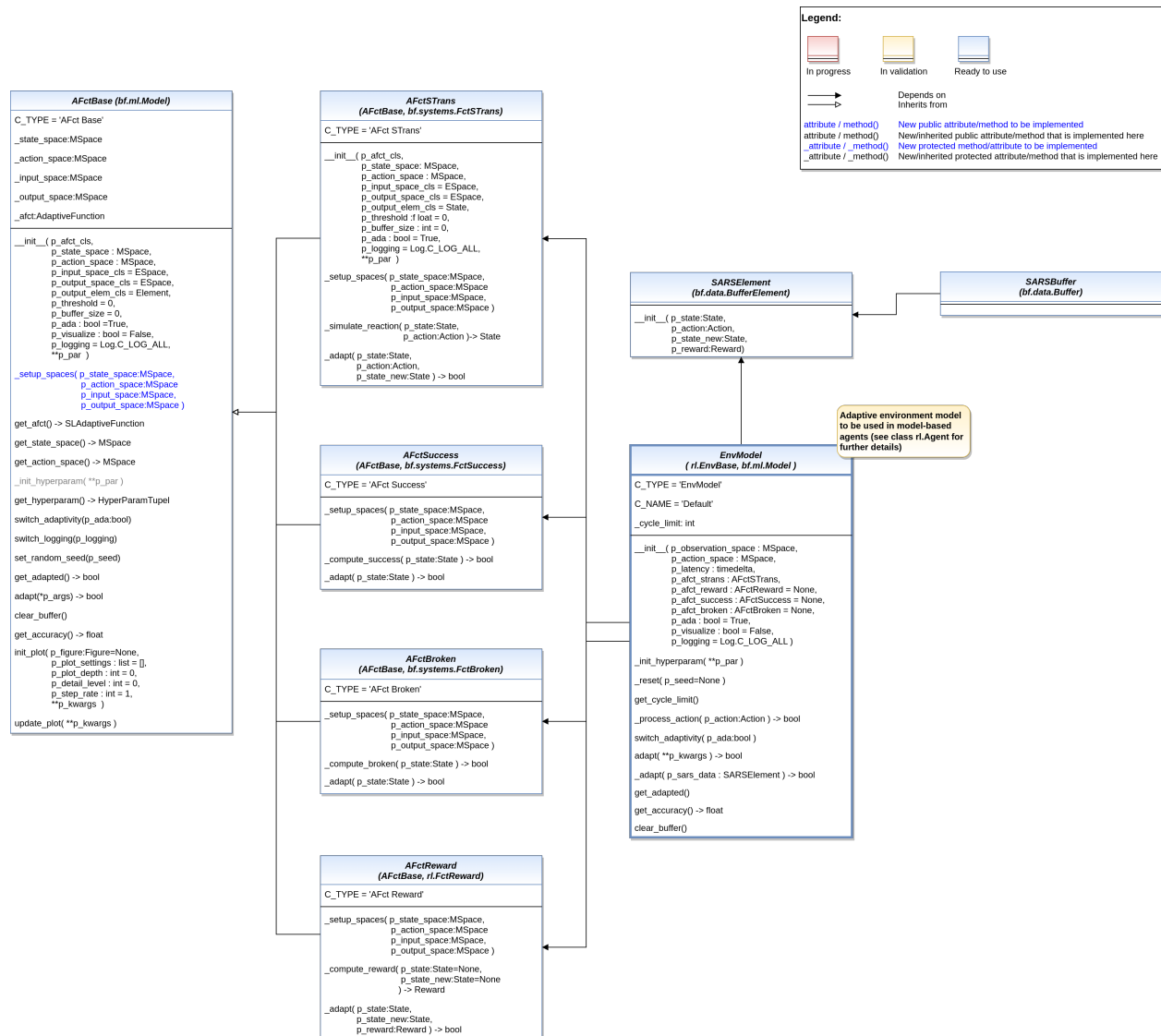
**Returns**

- **state\_space** (*MSpace*) – State space object
- **action\_space** (*MSpace*) – Action space object

**get\_cycle\_limit()** → int

Returns limit of cycles per training episode.

## RL-ENV-ADA - Environment Models



Ver. 1.1.0 (2023-04-03)

This module provides model classes for adaptive environment models.

```
class mlpro.rl.models_env_ada.AFctReward(p_afct_cls, p_state_space: ~mlpro.bf.math.basics.MSpace,
                                         p_action_space: ~mlpro.bf.math.basics.MSpace,
                                         p_input_space_cls=<class 'mlpro.bf.math.basics.ESpace'>,
                                         p_output_space_cls=<class 'mlpro.bf.math.basics.ESpace'>,
                                         p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>,
                                         p_threshold=0, p_buffer_size=0, p_ada: bool = True,
                                         p_visualize: bool = False, p_logging=True, **p_kwargs)
```

Bases: [AFctBase](#), [FctReward](#)

Online adaptive version of a reward function. See parent classes for further details.

**C\_TYPE** = 'AFct Reward'

```
_setup_spaces(p_state_space: MSpace, p_action_space: MSpace, p_input_space: MSpace,
               p_output_space: MSpace)
```

Custom method to set up the input and output space of the embedded adaptive function. Use the method `add_dimension()` of the empty spaces `p_input_space` and `p_output_space` to enrich them with suitable dimensions.

#### Parameters

- **p\_state\_space** ([MSpace](#)) – State space of an environment respectively observation space of an agent.
- **p\_action\_space** ([MSpace](#)) – Action space of an environment or agent.
- **p\_input\_space** ([MSpace](#)) – Empty input space of embedded adaptive function to be enriched with dimension.
- **p\_output\_space** ([MSpace](#)) – Empty output space of embedded adaptive function to be enriched with dimension.

```
_compute_reward(p_state_old: State = None, p_state_new: State = None) → Reward
```

Custom reward method. See method `compute_reward()` for further details.

```
_adapt(p_state: State, p_state_new: State, p_reward: Reward) → bool
```

Triggers adaptation of the embedded adaptive function.

#### Parameters

- **p\_state** ([State](#)) – Previous state.
- **p\_state\_new** ([State](#)) – New state.
- **p\_reward** ([Reward](#)) – Setpoint reward.

#### Returns

**adapted** – True, if something was adapted. False otherwise.

#### Return type

bool

```
class mlpro.rl.models_env_ada.SARSElement(p_state: State, p_action: Action, p_reward: Reward,
                                         p_state_new: State)
```

Bases: [BufferElement](#)

Element of a SARSBuffer.

```
class mlpro.rl.models_env_ada.SARSBuffer(p_size=1)
```

Bases: Buffer

State-Action-Reward-State-Buffer in dictionary.

```
class mlpro.rl.models_env_ada.EnvModel(p_observation_space: MSpace, p_action_space: MSpace,
                                       p_latency: timedelta, p_afct_strans: AFctSTrans, p_afct_reward:
                                       AFctReward = None, p_afct_success: AFctSuccess = None,
                                       p_afct_broken: AFctBroken = None, p_ada: bool = True,
                                       p_init_states: State = None, p_visualize: bool = False,
                                       p_logging=True)
```

Bases: [EnvBase](#), [Model](#)

Environment model class as part of a model-based agent.

#### Parameters

- **p\_observation\_space** ([MSpace](#)) – Observation space of related agent.
- **p\_action\_space** ([MSpace](#)) – Action space of related agent.
- **p\_latency** ([timedelta](#)) – Latency of related environment.
- **p\_afct\_strans** ([AFctSTrans](#)) – Mandatory external adaptive function for state transition.
- **p\_afct\_reward** ([AFctReward](#)) – Optional external adaptive function for reward computation.
- **p\_afct\_success** ([AFctSuccess](#)) – Optional external adaptive function for state assessment 'success'.
- **p\_afct\_broken** ([AFctBroken](#)) – Optional external adaptive function for state assessment 'broken'.
- **p\_ada** (*bool*) – Boolean switch for adaptivity.
- **p\_init\_states** ([State](#)) – Initial state of the env models.
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see class Log for more details).

```
C_TYPE = 'EnvModel'
```

```
C_NAME = 'Default'
```

```
static setup_spaces()
```

Static template method to set up and return state and action space of environment.

#### Returns

- **state\_space** ([MSpace](#)) – State space object
- **action\_space** ([MSpace](#)) – Action space object

```
_init_hyperparam(**p_par)
```

Implementation specific hyperparameters can be added here. Please follow these steps: a) Add each hyperparameter as an object of type HyperParam to the internal hyperparameter

space object self.\_hyperparam\_space

b) Create hyperparameter tuple and bind to self.\_hyperparam\_tuple

c) Set default value for each hyperparameter

**Parameters**

**p\_par** (*Dict*) – Further model specific hyperparameters, that are passed through constructor.

**\_reset**(*p\_seed=None*)

Custom method to reset the system to an initial/defined state. Use method `_set_status()` to set the state.

**Parameters**

**p\_seed** (*int*) – Seed parameter for an internal random generator

**get\_cycle\_limit**() → *int*

Returns limit of cycles per training episode.

**\_process\_action**(*p\_action: Action*) → *bool*

Custom method for state transition. To be implemented in a child class. See method `process_action()` for further details.

**switch\_adaptivity**(*p\_ada: bool*)

Switches adaption functionality on/off.

**Parameters**

**p\_ada** (*bool*) – Boolean switch for adaptivity

**adapt**(\*\**p\_kwargs*) → *bool*

Reactivated adaptation mechanism. See method `Model.adapt()` for further details.

**\_adapt**(*p\_sars\_elem: SARSElement*) → *bool*

Adapts the environment model based on State-Action-Reward-State (SARS) data.

**Parameters**

**p\_sars\_elem** (*SARSElement*) – Object of type `SARSElement`.

**get\_adapted**() → *bool*

Returns True, if the model was adapted at least once. False otherwise.

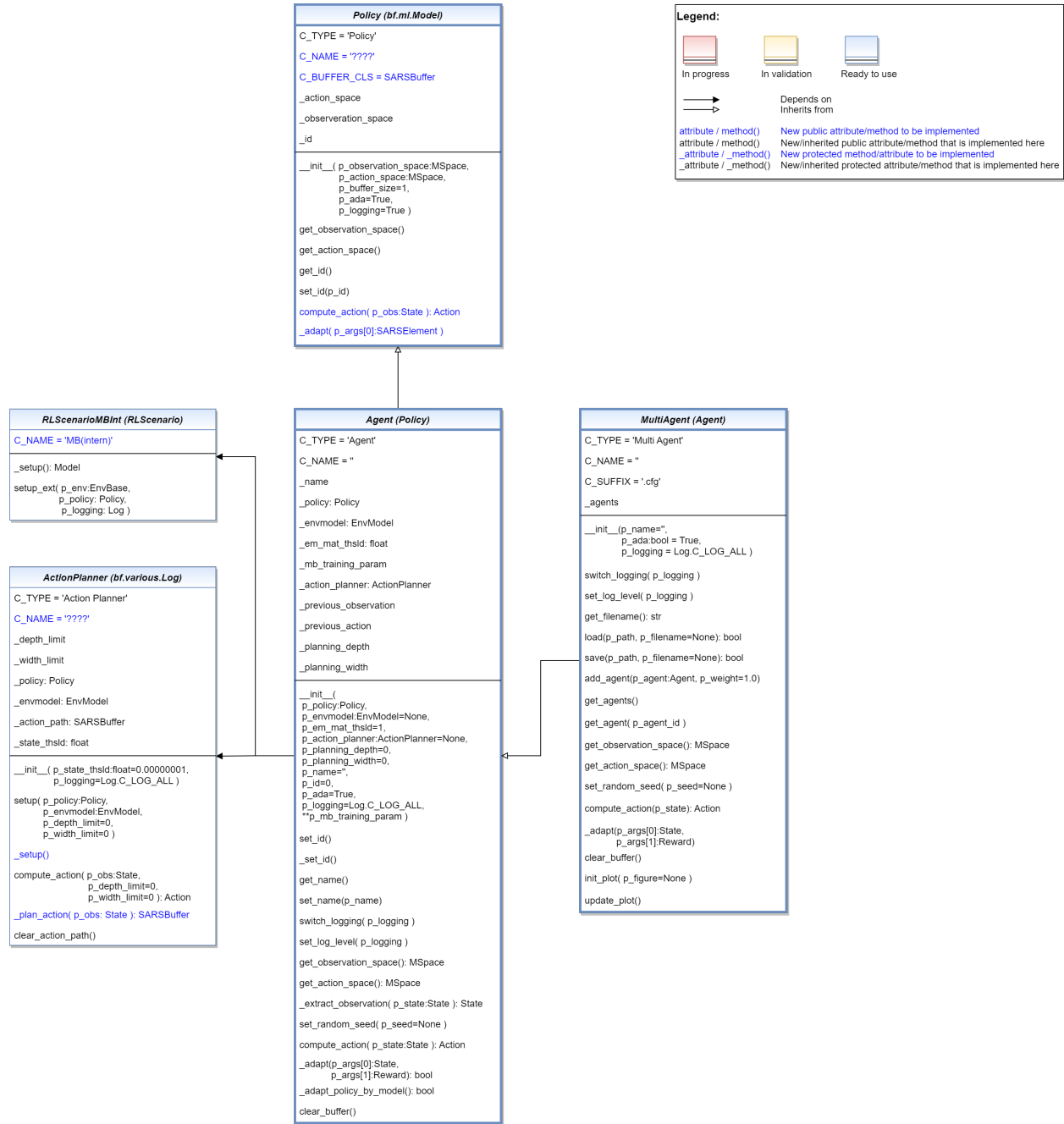
**get\_accuracy**()

Returns accuracy of environment model as average accuracy of the embedded adaptive functions.

**clear\_buffer**()

Clears internal buffer (if buffering is active).

## RL-AGENTS - Agents



Ver. 1.7.0 (2023-03-27)

This module provides model classes for policies, model-free and model-based agents and multi-agents.

**class** `mlpro.rl.models_agents.Policy`(*p\_observation\_space*: `MSpace`, *p\_action\_space*: `MSpace`, *p\_id*=None, *p\_buffer\_size*: `int` = 1, *p\_ada*: `bool` = True, *p\_visualize*: `bool` = False, *p\_logging*=True)

Bases: `Model`

This class represents the policy of a single-agent. It is adaptive and can be trained with State-Action-Reward (SAR) data that will be expected as a SAR buffer object. The three main learning paradigms of machine learning



to train a policy are supported:

- a) Training by Supervised Learning The entire SAR data set inside the SAR buffer shall be adapted.
- b) Training by Reinforcement Learning The latest SAR data record inside the SAR buffer shall be adapted.
- c) Training by Unsupervised Learning All state data inside the SAR buffer shall be adapted.

Furthermore, a policy class can compute actions from states.

Hyperparameters of the policy should be stored in the internal object `self._hp_list`, so that they can be tuned from outside. Optionally a policy-specific callback method can be called on changes. For more information see class `HyperParameterList`.

#### Parameters

- **p\_observation\_space** (`MSpace`) – Subspace of an environment that is observed by the policy.
- **p\_action\_space** (`MSpace`) – Action space object.
- **p\_id** – Optional external id
- **p\_buffer\_size** (`int`) – Size of internal buffer. Default = 1.
- **p\_ada** (`bool`) – Boolean switch for adaptivity. Default = True.
- **p\_visualize** (`bool`) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see constants of class `Log`). Default = `Log.C_LOG_ALL`.

**C\_TYPE** = 'Policy'

**C\_NAME** = '????'

**C\_BUFFER\_CLS**

alias of `SARSBuffer`

**get\_observation\_space()** → `MSpace`

**get\_action\_space()** → `MSpace`

**compute\_action**(`p_obs: State`) → `Action`

Specific action computation method to be redefined.

#### Parameters

**p\_obs** (`State`) – Observation data.

#### Returns

**action** – Action object.

#### Return type

`Action`

**\_adapt**(`p_sars_elem: SARSElement`) → bool

Adapts the policy based on State-Action-Reward-State (SARS) data.

#### Parameters

**p\_sars\_elem** (`SARSElement`) – Object of type `SARSElement`.

#### Returns

**adapted** – True, if something has been adapted. False otherwise.

#### Return type

bool

```
class mlpro.rl.models_agents.ActionPlanner(p_state_thsld=1e-08, p_logging=True)
```

Bases: [Log](#)

Template class for action planning algorithms to be used as part of model-based planning agents. The goal is to find the shortest sequence of actions that leads to a maximum reward.

#### Parameters

- **p\_state\_thsld** (*float*) – Threshold for metric difference between two states to be equal. Default = 0.00000001.
- **p\_logging** – Log level (see constants of class [Log](#)). Default = [Log.C\\_LOG\\_ALL](#).

**C\_TYPE** = 'Action Planner'

```
setup(p_policy: Policy, p_envmodel: EnvModel, p_prediction_horizon=0, p_control_horizon=0,
      p_width_limit=0)
```

Setup of action planner object in concrete planning scenario. Must be called before first planning. Optional custom method `_setup()` is called at the end.

#### Parameters

- **p\_policy** ([Policy](#)) – Policy of an agent.
- **p\_envmodel** ([EnvModel](#)) – Environment model.
- **p\_prediction\_horizon** (*int*) – Optional static maximum planning depth (=length of action path to be predicted). Can be overridden by method `compute_action()`. Default=0.
- **p\_control\_horizon** (*int*) – The length of predicted action path to be applied. Can be overridden by method `compute_action()`. Default=0.
- **p\_width\_limit** (*int*) – Optional static maximum planning width (=number of alternative actions per planning level). Can be overridden by method `compute_action()`. Default=0.

**\_setup()**

Optional custom setup method.

```
compute_action(p_obs: State, p_prediction_horizon=0, p_control_horizon=0, p_width_limit=0) → Action
```

Computes a path of actions with defined length that maximizes the reward of the given environment model. The planning algorithm itself is to be implemented in the custom method `_plan_action()`.

#### Parameters

- **p\_obs** ([State](#)) – Observation data.
- **p\_prediction\_horizon** (*int*) – Optional dynamic maximum planning depth (=length of action path to be predicted) that overrides the static limit of method `setup()`. Default=0 (no override).
- **p\_control\_horizon** (*int*) – The length of predicted action path to be applied that overrides the static limit of method `setup()`. Default=0 (no override).
- **p\_width\_limit** (*int*) – Optional dynamic maximum planning width (=number of alternative actions per planning level) that overrides the static limit of method `setup()`. Default=0 (no override).

#### Returns

**action** – Best action as result of the planning process.

#### Return type

[Action](#)

**`_plan_action(p_obs: State) → SARSBuffer`**

Custom planning algorithm to fill the internal action path (`self._action_path`). Search width and depth are restricted by the attributes `self._width_limit` and `self._prediction_horizon`.

**Parameters**

**`p_obs (State)`** – Observation data.

**Returns**

**`action_path`** – Sequence of `SARSElement` objects with included actions that lead to the best possible reward.

**Return type**

*SARSBuffer*

**`clear_action_path()`**

**`class mlpro.rl.models_agents.RLScenarioMBInt(p_mode=0, p_ada: bool = True, p_cycle_limit=0, p_visualize: bool = True, p_logging=True)`**

Bases: *RLScenario*

Internal use in class `Agent`. Intended for the training of the policy with the environment model of a model-based (single) agent.

**`C_NAME = 'MB(intern)'`**

**`_setup(**p_kwargs) → Model`**

Custom method to set up the ML scenario. Please bind your environment to `self._env` and return the agent as model.

**Parameters**

- **`p_mode`** – Operation mode. See `Mode.C_VALID_MODES` for valid values. Default = `Mode.C_MODE_SIM`
- **`p_ada (bool)`** – Boolean switch for adaptivity.
- **`p_visualize (bool)`** – Boolean switch for env/agent visualisation. Default = `True`.
- **`p_logging`** – Log level (see constants of class `Log`).

**Returns**

**`agent`** – Agent model (object of type `Agent` or `Multi-agent`)

**Return type**

*Agent*

**`setup_ext(p_env: EnvBase, p_policy: Policy, p_logging: Log)`**

**`class mlpro.rl.models_agents.Agent(p_policy: Policy, p_envmodel: EnvModel = None, p_em_acc_thsld=0.9, p_action_planner: ActionPlanner = None, p_predicting_horizon=0, p_controlling_horizon=0, p_planning_width=0, p_name="", p_ada=True, p_visualize: bool = True, p_logging=True, **p_mb_training_param)`**

Bases: *Policy*

This class represents a single agent model.

**Parameters**

- **`p_policy (Policy)`** – Policy object.
- **`p_envmodel (EnvModel)`** – Optional environment model object. Default = `None`.

- **p\_em\_acc\_thsld** (*float*) – Optional threshold for environment model accuracy (whether the envmodel is ‘good’ enough to be used to train the policy). Default = 0.9.
- **p\_action\_planner** (*ActionPlanner*) – Optional action planner object (obligatory for model based agents). Default = None.
- **p\_predicting\_horizon** (*int*) – Optional predicting horizon (obligatory for model based agents). Default = 0.
- **p\_controlling\_horizon** (*int*) – Optional controlling (obligatory for model based agents). Default = 0.
- **p\_planning\_width** (*int*) – Optional planning width (obligatory for model based agents). Default = 0.
- **p\_name** (*str*) – Optional name of agent. Default = ‘’.
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C\_LOG\_ALL.
- **p\_mb\_training\_param** (*dict*) – Optional parameters for internal policy training with environment model (see parameters of class RLTraining). Hyperparameter tuning and data logging is not supported here. The suitable scenario class is added internally.

**C\_TYPE** = 'Agent'

**C\_NAME** = ''

**\_init\_hyperparam**(\*\*p\_par)

Implementation specific hyperparameters can be added here. Please follow these steps: a) Add each hyperparameter as an object of type HyperParam to the internal hyperparameter

space object self.\_hyperparam\_space

b) Create hyperparameter tuple and bind to self.\_hyperparam\_tuple

c) Set default value for each hyperparameter

**Parameters**

**p\_par** (*Dict*) – Further model specific hyperparameters, that are passed through constructor.

**switch\_logging**(p\_logging)

Sets new log level.

**Parameters**

**p\_logging** – Log level (constant C\_LOG\_LEVELS contains valid values)

**switch\_adaptivity**(p\_ada: *bool*)

Switches adaption functionality on/off.

**Parameters**

**p\_ada** (*bool*) – Boolean switch for adaptivity

**set\_log\_level**(p\_level)

**get\_observation\_space**() → *MSpace*

**get\_action\_space()** → *MSpace*

**\_extract\_observation**(*p\_state*: *State*) → *State*

**set\_random\_seed**(*p\_seed*=None)

Resets the internal random generator using the given seed.

**compute\_action**(*p\_state*: *State*) → *Action*

Default implementation of a single agent.

**Parameters**

**p\_state** (*State*) – State of the related environment.

**Returns**

**action** – Action object.

**Return type**

*Action*

**\_adapt**(*p\_state*: *State*, *p\_reward*: *Reward*) → bool

Default adaptation implementation of a single agent.

**Parameters**

- **p\_state** (*State*) – State object.
- **p\_reward** (*Reward*) – Reward object.

**Returns**

**result** – True, if something has been adapted. False otherwise.

**Return type**

bool

**\_adapt\_policy\_by\_model**()

**clear\_buffer**()

Clears internal buffer (if buffering is active).

**class** mlpro.rl.models\_agents.**MultiAgent**(*p\_name*: str = "", *p\_ada*: bool = True, *p\_visualize*: bool = False, *p\_logging*=True)

Bases: *Agent*

Multi-Agent.

**Parameters**

- **p\_name** (str) – Name of agent. Default = ''.
- **p\_ada** (bool) – Boolean switch for adaptivity. Default = True.
- **p\_visualize** (bool) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default = Log.C\_LOG\_ALL.

**C\_TYPE** = 'Multi-Agent'

**C\_NAME** = ''

**switch\_logging**(*p\_logging*) → None

Sets new log level.

**Parameters**

**p\_logging** – Log level (constant C\_LOG\_LEVELS contains valid values)

**switch\_adaptivity**(*p\_ada*: *bool*)

Switches adaption functionality on/off.

**Parameters**

**p\_ada** (*bool*) – Boolean switch for adaptivity

**set\_log\_level**(*p\_level*)

**add\_agent**(*p\_agent*: *Agent*, *p\_weight*=1.0) → None

Adds agent object to internal list of agents.

**Parameters**

- **p\_agent** (*Agent*) – Agent object to be added.
- **p\_weight** (*float*) – Optional weight for the agent. Default = 1.0.

**get\_agents**()

**get\_agent**(*p\_agent\_id*)

Returns information of a single agent.

**Returns**

**agent\_info** – agent\_info[0] is the agent object itself and agent\_info[1] it's weight

**Return type**

tuple

**get\_observation\_space**() → *MSpace*

**get\_action\_space**() → *MSpace*

**set\_random\_seed**(*p\_seed*=None)

Resets the internal random generator using the given seed.

**compute\_action**(*p\_state*: *State*) → *Action*

Default implementation of a single agent.

**Parameters**

**p\_state** (*State*) – State of the related environment.

**Returns**

**action** – Action object.

**Return type**

*Action*

**\_adapt**(*p\_state*: *State*, *p\_reward*: *Reward*) → bool

Default adaptation implementation of a single agent.

**Parameters**

- **p\_state** (*State*) – State object.
- **p\_reward** (*Reward*) – Reward object.

**Returns**

**result** – True, if something has been adapted. False otherwise.

**Return type**

bool

**clear\_buffer()**

Clears internal buffer (if buffering is active).

**init\_plot**(*p\_figure*: Figure = None, *p\_plot\_settings*: list = Ellipsis, *p\_plot\_depth*: int = 0, *p\_detail\_level*: int = 0, *p\_step\_rate*: int = 0, **\*\*p\_kwargs**)

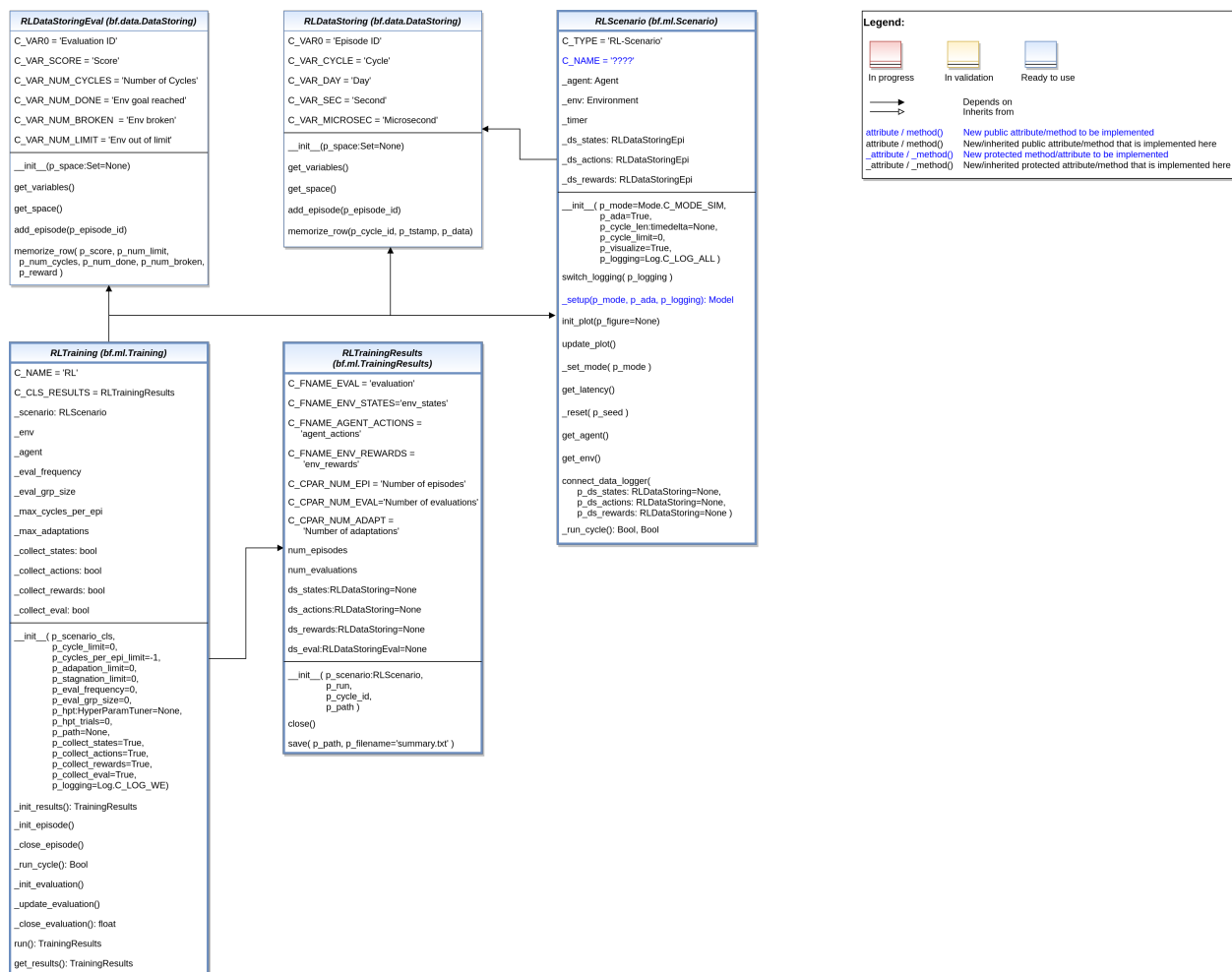
Doesn't support embedded plot of underlying agent hierarchy.

**update\_plot**(**\*\*p\_kwargs**)

Updates the plot.

**Parameters**

**\*\*p\_kwargs** – Implementation-specific plot data and/or parameters.

**RL-TRAIN - Scenarios, Training and Tuning**

Ver. 2.0.1 (2023-09-25)

This module provides model classes to define and run rl scenarios and to train agents inside them.

**class** mlpro.rl.models\_train.**RLDataStoring**(*p\_space*: Set = None)

Bases: DataStoring

Derivative of basic class DataStoring that is specialized to store episodic training data in the context of reinforcement learning.

**Parameters**

**p\_space** ([Set](#)) – Space object that provides dimensional information for raw data. If None, a training header data object will be instantiated.

**C\_VAR0** = 'Episode ID'

**C\_VAR\_CYCLE** = 'Cycle'

**C\_VAR\_DAY** = 'Day'

**C\_VAR\_SEC** = 'Second'

**C\_VAR\_MICROSEC** = 'Microsecond'

**get\_variables()**

**get\_space()**

**add\_episode**(*p\_episode\_id*)

**memorize\_row**(*p\_cycle\_id*, *p\_tstamp*: *timedelta*, *p\_data*)

Memorizes an episodic data row.

**Parameters**

- **id** (*p\_cycle\_id* *Cycle*)
- **stamp** (*p\_tstamp* *Time*)
- **space** (*p\_data* *Data that meet the dimensionality of the related*)

**class** mlpro.rl.models\_train.**RLDataStoringEval**(*p\_space*: [Set](#))

Bases: [DataStoring](#)

Derivative of basic class [DataStoring](#) that is specialized to store evaluation data of a training in the context of reinforcement learning.

**Parameters**

- **p\_space** ([Set](#)) – Set object that provides dimensional information for raw data. If None a training header data object will
- **instantiated.** (*be*)

**C\_VAR0** = 'Evaluation ID'

**C\_VAR\_SCORE** = 'Score'

**C\_VAR\_SCORE\_MA** = 'Score(MA)'

**C\_VAR\_SCORE\_UNTIL\_STAG** = 'Score until Stagnation'

**C\_VAR\_SCORE\_MA\_UNTIL\_STAG** = 'Score(MA) until Stagnation'

**C\_VAR\_NUM\_CYCLES** = 'Cycles'

**C\_VAR\_NUM\_SUCCESS** = 'Successes'

**C\_VAR\_NUM\_BROKEN** = 'Crashes'

**C\_VAR\_NUM\_LIMIT** = 'Timeouts'



**C\_VAR\_NUM\_ADAPT** = 'Adaptations'

**get\_variables()**

**get\_space()**

**add\_evaluation**(*p\_evaluation\_id*)

**memorize\_row**(*p\_score*, *p\_score\_ma*, *p\_num\_limit*, *p\_num\_cycles*, *p\_num\_success*, *p\_num\_broken*,  
*p\_num\_adaptations*, *p\_reward*, *p\_score\_until\_stag=None*, *p\_score\_ma\_until\_stag=None*)

Memorizes an evaluation data row.

#### Parameters

- **p\_score** (*float*) – Score value of current evaluation.
- **p\_score\_ma** (*float*) – Moving average score value.
- **p\_num\_limit** (*int*) – Number of episodes in timeout.
- **p\_num\_cycles** (*int*) – Number of evaluation cycles.
- **p\_num\_success** (*int*) – Number of states that were labeled as successfully.
- **p\_num\_broken** (*int*) – Number of states that were labeled as broken.
- **p\_num\_adaptations** (*int*) – Number of adaptations in the last training period.
- **p\_reward** (*list*) – Episode Reward
- **p\_score\_until\_stag** (*float*) – Optional score value of current evaluation until first stagnation. Default = None.
- **p\_score\_ma\_until\_stag** (*float*) – Optional moving average score value until first stagnation. Default = None.

**class** mlpro.rl.models\_train.RLScenario(*p\_mode=0*, *p\_ada: bool = True*, *p\_cycle\_limit=0*, *p\_visualize: bool = True*, *p\_logging=True*)

Bases: [Scenario](#)

Template class for an RL scenario consisting of an environment and an agent.

#### Parameters

- **p\_mode** – Operation mode. See `bf.ops.Mode.C_VALID_MODES` for valid values. Default = `Mode.C_MODE_SIM`.
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = `True`.
- **p\_cycle\_limit** (*int*) – Maximum number of cycles (0=no limit, -1=get from env). Default = 0.
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = `False`.
- **p\_logging** – Log level (see constants of class `mlpro.bf.various.Log`). Default = `Log.C_LOG_WE`.

#### C\_TYPE

Constant class type for logging: 'RL-Scenario'.

#### Type

str

**C\_NAME**

Constant custom name for logging. To be set in own child class.

**Type**

str

**C\_TYPE** = 'RL-Scenario'

**C\_NAME** = '????'

**\_reduce\_state**(*p\_state: dict, p\_path: str, p\_os\_sep: str, p\_filename\_stub: str*)

Custom method to reduce the given object state by components that can not be pickled. Further data files can be created in the given path and should use the given filename stub.

**Parameters**

- **p\_state** (*dict*) – Object state dictionary to be reduced by components that can not be pickled.
- **p\_path** (*str*) – Path to store further optional custom data files
- **p\_os\_sep** (*str*) – OS-specific path separator.
- **p\_filename\_stub** (*str*) – Filename stub to be used for further optional custom data files

**\_complete\_state**(*p\_path: str, p\_os\_sep: str, p\_filename\_stub: str*)

Custom method to complete the object state (=self) from external data sources. This method is called by standard method `__setstate__()` during unpickling the object from an external file.

**Parameters**

- **p\_path** (*str*) – Path of the object pickle file (and further optional related files)
- **p\_os\_sep** (*str*) – OS-specific path separator.
- **p\_filename\_stub** (*str*) – Filename stub to be used for further optional custom data files

**switch\_logging**(*p\_logging*)

Sets new log level.

**Parameters**

**p\_logging** – Log level (constant `C_LOG_LEVELS` contains valid values)

**\_setup**(*p\_mode, p\_ada: bool, p\_visualize: bool, p\_logging*) → *Model*

Custom method to set up the ML scenario. Please bind your environment to `self._env` and return the agent as model.

**Parameters**

- **p\_mode** – Operation mode. See `Mode.C_VALID_MODES` for valid values. Default = `Mode.C_MODE_SIM`
- **p\_ada** (*bool*) – Boolean switch for adaptivity.
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = True.
- **p\_logging** – Log level (see constants of class Log).

**Returns**

**agent** – Agent model (object of type Agent or Multi-agent)

**Return type**

*Agent*

**init\_plot**(*p\_figure*: *Figure = None*, *p\_plot\_settings*: *PlotSettings = None*)

Initializes the plot functionalities of the class.

#### Parameters

- **p\_figure** (*Matplotlib.figure.Figure*, *optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p\_plot\_settings** (*PlotSettings*) – Optional plot settings. If None, the default view is plotted (see attribute C\_PLOT\_DEFAULT\_VIEW).

**update\_plot**(\*\**p\_kwargs*)

Updates the plot.

#### Parameters

**\*\*p\_kwargs** – Implementation-specific plot data and/or parameters.

**\_set\_mode**(*p\_mode*)

Custom method to set the operation mode of components of the scenario. See method `set_mode()` for further details.

### Parameter

#### **p\_mode**

Operation mode. See class `bf.ops.Mode` for further details.

**get\_latency**() → *timedelta*

Returns the latency of the scenario. To be implemented in child class.

**\_reset**(*p\_seed*)

Environment and timer are reset. The random generators for environment and agent will also be reset. Optionally the agent's internal buffer data will be cleared, but it's policy will not be touched.

#### Parameters

**generator** (*p\_seed* *New seed for environment's and agent's random*)

**get\_agent**()

**get\_env**()

**connect\_data\_logger**(*p\_ds\_states*: *RLDataStoring = None*, *p\_ds\_actions*: *RLDataStoring = None*,  
*p\_ds\_rewards*: *RLDataStoring = None*)

**\_run\_cycle**()

Processes a single cycle.

#### Returns

- **success** (*bool*) – True on success. False otherwise.
- **error** (*bool*) – True on error. False otherwise.
- **adapted** (*bool*) – True, if agent adapted something in this cycle. False otherwise.
- **end\_of\_data** (*bool*) – True, if the end of the related data source has been reached. False otherwise.

```
class mlpro.rl.models_train.RLTrainingResults(p_scenario: RLScenario, p_run, p_cycle_id,
                                              p_logging='W')
```

Bases: [TrainingResults](#)

Results of a RL training.

#### Parameters

- **p\_scenario** ([RLScenario](#)) – Related reinforcement learning scenario.
- **p\_run** (*int*) – Run id.
- **p\_cycle\_id** (*int*) – Id of first cycle of this run.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL

**C\_NAME** = 'RL'

**C\_FNAME\_EVAL** = 'evaluation'

**C\_FNAME\_ENV\_STATES** = 'env\_states'

**C\_FNAME\_AGENT\_ACTIONS** = 'agent\_actions'

**C\_FNAME\_ENV\_REWARDS** = 'env\_rewards'

**C\_CPAR\_NUM\_EPI** = 'Training Episodes'

**C\_CPAR\_NUM\_EVAL** = 'Evaluations'

**close()**

**\_log\_results()**

**save**(*p\_path*, *p\_filename*='summary.csv') → bool

Saves a training summary in the given path.

#### Parameters

- **p\_path** (*str*) – Destination folder
- **p\_filename** (*string*) – Name of summary file. Default = 'summary.csv'

#### Returns

**success** – True, if summary file was created successfully. False otherwise.

#### Return type

bool

```
class mlpro.rl.models_train.RLTraining(**p_kwargs)
```

Bases: [Training](#)

This class performs an episodic training on a (multi-)agent in a given environment. Both are expected as parts of a reinforcement learning scenario (see class [RLScenario](#) for more details). The class optionally collects all relevant data like environment states and rewards or agents actions. Furthermore, overarching training data will be collected.

#### Parameters

- **p\_scenario\_cls** – Name of RL scenario class, compatible to/inherited from class [RLScenario](#).
- **p\_cycle\_limit** (*int*) – Maximum number of training cycles (0=no limit). Default = 0.

- **p\_cycles\_per\_epi\_limit** (*int*) – Optional limit of cycles per episode (0=no limit, -1=get environment limit). Default = -1.
- **p\_adaptation\_limit** (*int*) – Maximum number of adaptations (0=no limit). Default = 0.
- **p\_eval\_frequency** (*int*) – Optional evaluation frequency (0=no evaluation). Default = 0.
- **p\_eval\_grp\_size** (*int*) – Number of evaluation episodes (eval group). Default = 0.
- **p\_score\_ma\_horizon** (*int*) – Horizon length for moving average score computation. Default = 5.
- **p\_stagnation\_limit** (*int*) – Optional limit of consecutive evaluations without training progress. Base is the moving average score. Default = 0.
- **p\_stagnation\_entry** (*int*) – Optional number of evaluations before the stagnation detection starts. Default = 0.
- **p\_end\_at\_stagnation** (*bool*) – If True, the training ends when stagnation has been detected. Default = True.
- **p\_hpt** ([HyperParamTuner](#)) – Optional hyperparameter tuner (see class `mlpro.bf.ml.HyperParamTuner`). Default = None.
- **p\_hpt\_trials** (*int*) – Optional number of hyperparameter tuning trials. Default = 0. Must be > 0 if `p_hpt` is supplied.
- **p\_path** (*str*) – Optional destination path to store training data. Default = None.
- **p\_collect\_states** (*bool*) – If True, the environment states will be collected. Default = True.
- **p\_collect\_actions** (*bool*) – If True, the agent actions will be collected. Default = True.
- **p\_collect\_rewards** (*bool*) – If True, the environment reward will be collected. Default = True.
- **p\_collect\_eval** (*bool*) – If True, global evaluation data will be collected. Default = True.
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see constants of class `mlpro.bf.various.Log`). Default = `Log.C_LOG_WE`.

**C\_NAME** = 'RL'

**C\_CLS\_RESULTS**

alias of [RLTrainingResults](#)

**\_init\_results()** → [TrainingResults](#)

**\_init\_episode()**

**\_close\_episode()**

**\_init\_evaluation()**

Initializes the next evaluation.

**\_update\_evaluation**(*p\_success: bool, p\_error: bool, p\_cycle\_limit: bool*)

Updates evaluation statistics.

#### Parameters

- **p\_success** (*bool*) – True on success. False otherwise.

- **p\_error** (*bool*) – True on error. False otherwise.
- **p\_cycle\_limit** (*bool*) – True, if cycle limit has reached. False otherwise.

**\_close\_evaluation()** → ndarray

Closes the current evaluation and computes a related score.

**Returns**

**score** – Score of current evaluation.

**Return type**

float

**\_run\_cycle()** → bool

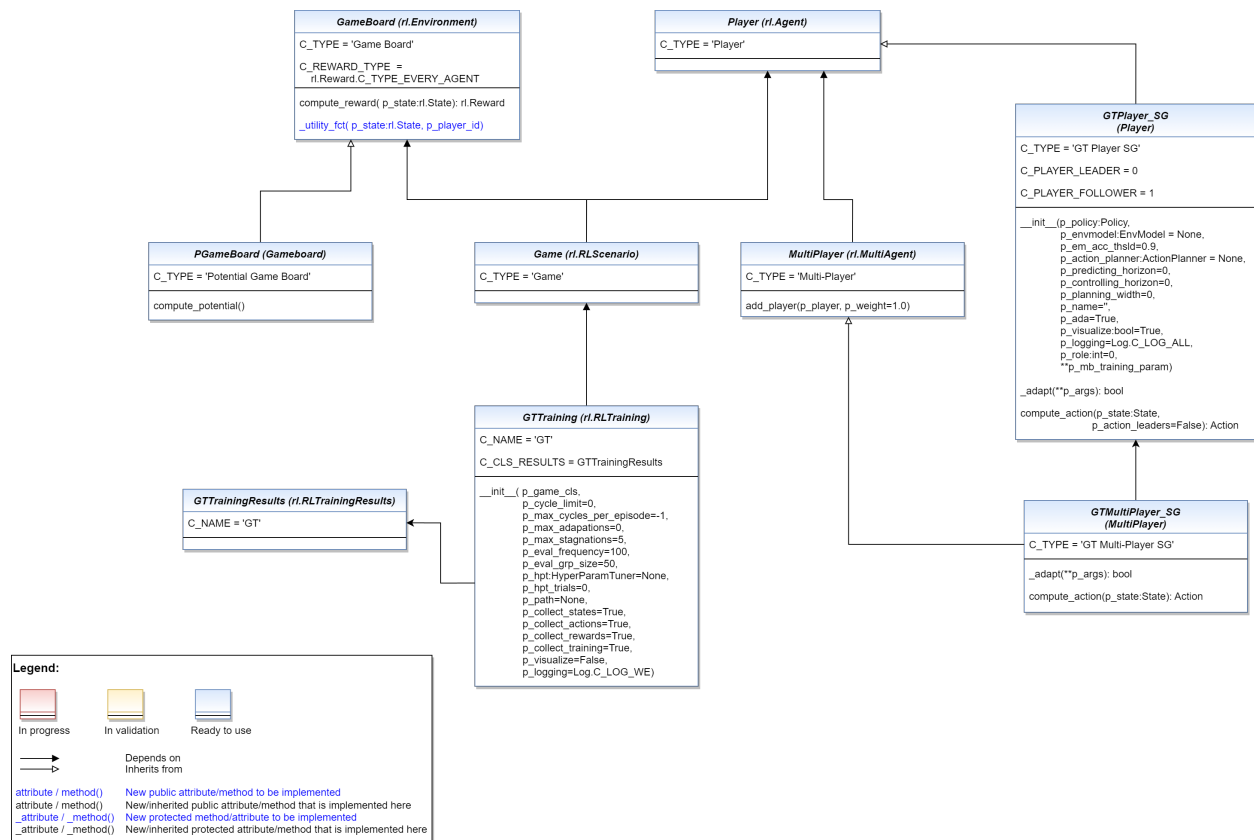
Runs single training cycle.

**Returns**

True, if training has finished. False otherwise.

## 11.1.4 MLPro-GT - Game Theory

### Dynamic Games



Ver. 2.3.0 (2023-09-25)

This module provides model classes for tasks related to Game Theory in dynamic games.

```
class mlpro.gt.dynamicgames.basics.GameBoard(p_mode=0, p_latency: timedelta = None, p_fct_strans:
    FctSTrans = None, p_fct_reward: FctReward = None,
    p_fct_success: FctSuccess = None, p_fct_broken:
    FctBroken = None, p_mujoco_file=None, p_frame_skip:
    int = 1, p_state_mapping=None,
    p_action_mapping=None, p_camera_conf: tuple =
    (None, None, None), p_visualize: bool = False,
    p_logging=True)
```

Bases: [Environment](#)

Model class for a game theoretical game board. See super class for more information.

**C\_TYPE** = 'Game Board'

**C\_REWARD\_TYPE** = 1

**\_compute\_reward**(p\_state\_old: [State](#), p\_state\_new: [State](#)) → [Reward](#)

Custom reward method. See method compute\_reward() for further details.

**\_utility\_fct**(p\_state: [State](#), p\_player\_id)

Computes utility of given player. To be redefined.

```
class mlpro.gt.dynamicgames.basics.Player(p_policy: Policy, p_envmodel: EnvModel = None,
    p_em_acc_thsld=0.9, p_action_planner: ActionPlanner =
    None, p_predicting_horizon=0, p_controlling_horizon=0,
    p_planning_width=0, p_name="", p_ada=True, p_visualize:
    bool = True, p_logging=True, **p_mb_training_param)
```

Bases: [Agent](#)

This class implements a game theoretical player model. See super class for more information.

**C\_TYPE** = 'Player'

```
class mlpro.gt.dynamicgames.basics.MultiPlayer(p_name: str = "", p_ada: bool = True, p_visualize:
    bool = False, p_logging=True)
```

Bases: [MultiAgent](#)

This class implements a game theoretical model for a team of players. See super class for more information.

**C\_TYPE** = 'Multi-Player'

**add\_player**(p\_player: [Player](#), p\_weight=1.0) → None

```
class mlpro.gt.dynamicgames.basics.Game(p_mode=0, p_ada: bool = True, p_cycle_limit=0, p_visualize:
    bool = True, p_logging=True)
```

Bases: [RLScenario](#)

This class implements a game consisting of a game board and a (multi-)player. See super class for more information.

**C\_TYPE** = 'Game'

```
class mlpro.gt.dynamicgames.basics.GTTrainingResults(p_scenario: RLScenario, p_run, p_cycle_id,
    p_logging='W')
```

Bases: [RLTrainingResults](#)

Results of a GT training.

**C\_NAME** = 'GT'

**class** mlpro.gt.dynamicgames.basics.**GTTraining**(\*\*p\_kwargs)

Bases: [RLTraining](#)

This class implements a standardized episodic training process. See super class for more information.

#### Parameters

- **p\_game\_cls** – Name of GT game class, compatible to/inherited from class Game.
- **p\_cycle\_limit** (*int*) – Maximum number of training cycles (0=no limit). Default = 0.
- **p\_cycles\_per\_epi\_limit** (*int*) – Optional limit of cycles per episode (0=no limit, -1=get environment limit). Default = -1.
- **p\_adaptation\_limit** (*int*) – Maximum number of adaptations (0=no limit). Default = 0.
- **p\_stagnation\_limit** (*int*) – Optional limit of consecutive evaluations without training progress. Default = 0.
- **p\_eval\_frequency** (*int*) – Optional evaluation frequency (0=no evaluation). Default = 0.
- **p\_eval\_grp\_size** (*int*) – Number of evaluation episodes (eval group). Default = 0.
- **p\_hpt** ([HyperParamTuner](#)) – Optional hyperparameter tuner (see class mlpro.bf.ml.HyperParamTuner). Default = None.
- **p\_hpt\_trials** (*int*) – Optional number of hyperparameter tuning trials. Default = 0. Must be > 0 if p\_hpt is supplied.
- **p\_path** (*str*) – Optional destination path to store training data. Default = None.
- **p\_collect\_states** (*bool*) – If True, the environment states will be collected. Default = True.
- **p\_collect\_actions** (*bool*) – If True, the agent actions will be collected. Default = True.
- **p\_collect\_rewards** (*bool*) – If True, the environment reward will be collected. Default = True.
- **p\_collect\_training** (*bool*) – If True, global training data will be collected. Default = True.
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C\_LOG\_WE.

**C\_NAME** = 'GT'

**C\_CLS\_RESULTS**

alias of [GTTrainingResults](#)

Ver. 1.1.0 (2023-05-11)

This module provides model classes for Potential Games in dynamic programming.

```
class mlpro.gt.dynamicgames.potential.PGameBoard(p_mode=0, p_latency: timedelta = None,
p_fct_strans: FctSTrans = None, p_fct_reward:
FctReward = None, p_fct_success: FctSuccess =
None, p_fct_broken: FctBroken = None,
p_mujoco_file=None, p_frame_skip: int = 1,
p_state_mapping=None, p_action_mapping=None,
p_camera_conf: tuple = (None, None, None),
p_visualize: bool = False, p_logging=True)
```



Bases: [GameBoard](#)

Model class for a potential game theoretical game board. See super class for more information.

**C\_TYPE = 'Potential Game Board'**

**compute\_potential()**

Computes (weighted) potential level of the game board.

Ver. 1.1.1 (2023-08-22)

This module provides model classes for Stackelberg Games in dynamic programming.

```
class mlpro.gt.dynamicgames.stackelberg.GTPlayer_SG(p_policy: Policy, p_envmodel: EnvModel =
None, p_em_acc_thsld=0.9, p_action_planner: ActionPlanner = None, p_predicting_horizon=0,
p_controlling_horizon=0, p_planning_width=0,
p_name="", p_ada=True, p_visualize: bool =
True, p_logging=True, p_role: int = 0,
**p_mb_training_param)
```

Bases: [Player](#)

This class implements a game theoretical player model in a stackelberg game mode, in which there is a possibility to assign the role of the player as a leader or follower.

The leader(s) has a priority to compute actions and adapt policies over the followers. Then, the followers can react according to the selected actions by the leaders, while computing their actions and adapting their policies. Thus, as followers, the selected actions by the leaders will assign as one of the inputs on both `_adapt` and `compute_action` methods.

#### Parameters

**p\_role** – Role of the player. Default = C\_PLAYER\_LEADER.

**C\_TYPE = 'GT Player SG'**

**C\_PLAYER\_LEADER = 0**

**C\_PLAYER\_FOLLOWER = 1**

**\_adapt(\*\*p\_args) → bool**

Default adaptation implementation of a single agent.

#### Parameters

- **p\_state** ([State](#)) – State object.
- **p\_reward** ([Reward](#)) – Reward object.

#### Returns

**result** – True, if something has been adapted. False otherwise.

#### Return type

bool

**compute\_action(p\_state: [State](#), p\_action\_leaders=False) → [Action](#)**

Default implementation of a single agent.

#### Parameters

**p\_state** ([State](#)) – State of the related environment.

#### Returns

**action** – Action object.

**Return type***Action*

```
class mlpro.gt.dynamicgames.stackelberg.GTMultiPlayer_SG(p_name: str = "", p_ada: bool = True,
                                                         p_visualize: bool = False,
                                                         p_logging=True)
```

Bases: *MultiPlayer*

This class implements a game theoretical multi-player model in a stackelberg game mode.

**C\_TYPE** = 'GT Multi-Player SG'**\_adapt**(\*\*p\_args) → bool

Default adaptation implementation of a single agent.

**Parameters**

- **p\_state** (*State*) – State object.
- **p\_reward** (*Reward*) – Reward object.

**Returns**

**result** – True, if something has been adapted. False otherwise.

**Return type**

bool

**compute\_action**(p\_state: *State*) → *Action*

Default implementation of a single agent.

**Parameters**

**p\_state** (*State*) – State of the related environment.

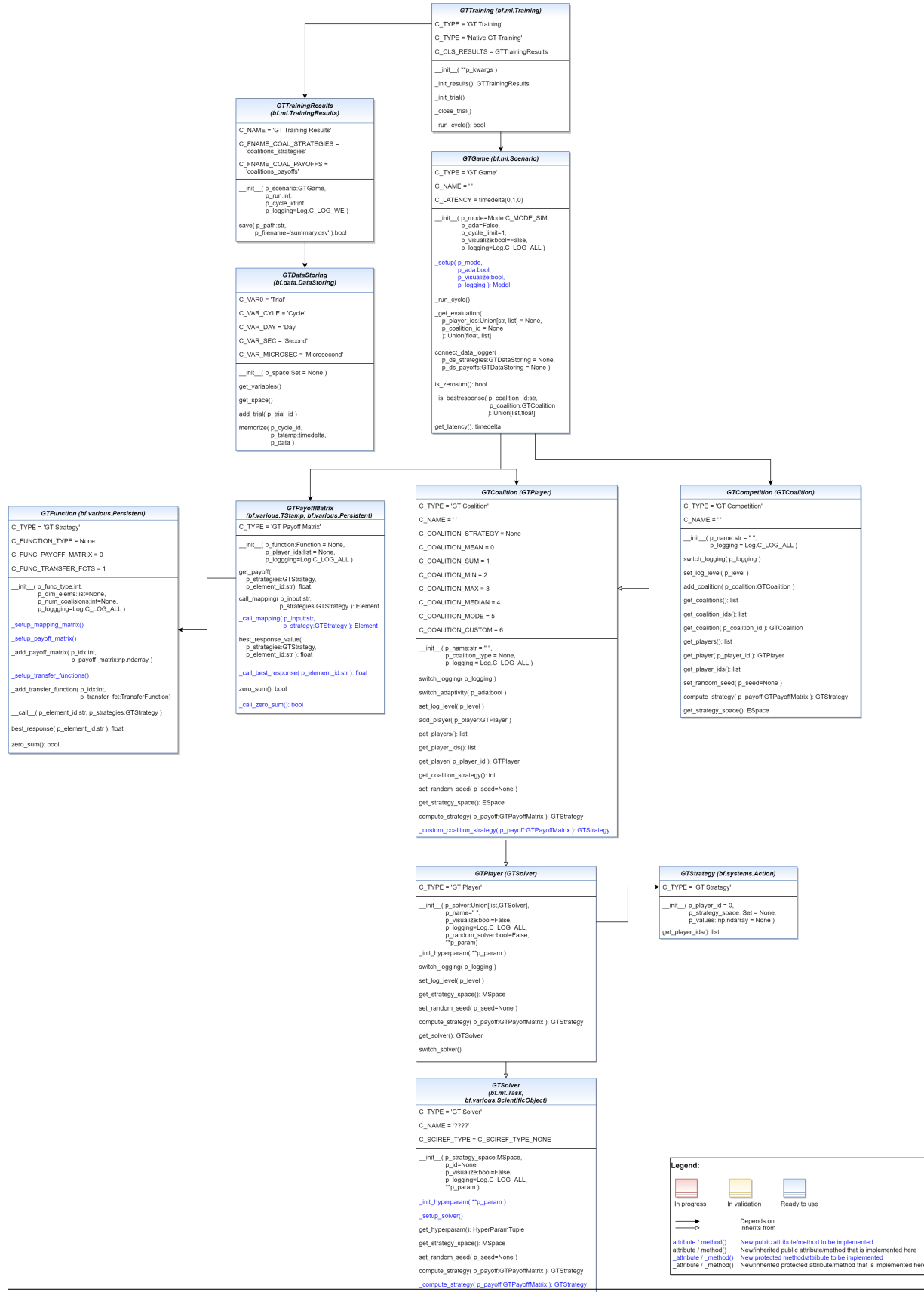
**Returns**

**action** – Action object.

**Return type***Action*



## Native GT



This module provides model classes for tasks related to a Native Game Theory.

```
class mlpro.gt.native.basics.GTStrategy(p_player_id=0, p_strategy_space: Set = None, p_values:
                                         ndarray = None)
```

Bases: *Action*

A class representing a strategy for a player in game theory. Objects of this class representations of (multi-)players. Every element of the internal list is related to a player, and its partial subsection. Strategy values for the first player can be added while object instantiation. Strategy values of further player can be added by using method `self.add_elem()`.

#### Parameters

- **p\_player\_id** – Unique id of (first) player to be added
- **p\_strategy\_space** (*Set*) – Strategy space of (first) player to be added. Default = None.
- **p\_values** (*np.ndarray*) – Strategy values of (first) player to be added. Default = None.

**C\_TYPE** = 'GT Strategy'

**get\_player\_ids()** → list

A method to get the ids of (multi-)players that have been added to this class.

#### Returns

A list of players' ids.

#### Return type

list

```
class mlpro.gt.native.basics.GTFunction(p_func_type: int, p_dim_elems: list = None, p_num_coalitions:
                                         int = None, p_logging=True)
```

Bases: *Persistent*

A class representing a mapping functionality between strategies and payoffs in two possible forms, such as payoff matrices or transfer functions.

#### Parameters

- **p\_func\_type** (*int*) – Type of functions, either `C_FUNC_PAYOFF_MATRIX` or `C_FUNC_TRANSFER_FCTS`.
- **p\_dim\_elems** (*list*) – The dimension of payoff matrix. For transfer function, this can be avoided. Default = None.
- **p\_num\_coalitions** (*int*) – Number of coalitions. For transfer function, this can be avoided. Default = None.
- **p\_logging** – Log level (see constants `C_LOG_*`). Default: `Log.C_LOG_ALL`.

**C\_TYPE** = 'GT Function'

**C\_FUNC\_PAYOFF\_MATRIX** = 0

**C\_FUNC\_TRANSFER\_FCTS** = 1

**C\_FUNCTION\_TYPE** = None

**\_setup\_mapping\_matrix()** → ndarray

A method to setup the mapping of the payoff matrix between strategies and payoffs. This is only applicable for `C_FUNC_PAYOFF_MATRIX`. This method needs to be redefined based on the setup of the game.

**Returns**

Resulted mapping.

**Return type**

np.ndarray

**`_setup_payoff_matrix()`**

A method to setup payoff matrices. This is only applicable for C\_FUNC\_PAYOFF\_MATRIX. This method needs to be redefined based on the setup of the game.

**`_add_payoff_matrix(p_idx: int, p_payoff_matrix: ndarray)`**

A method to add a payoff matrix to the GTFunction class. This method is called during the redifinition of `_setup_payoff_matrix()`. This is only applicable for C\_FUNC\_PAYOFF\_MATRIX.

**Parameters**

- **`p_idx (int)`** – Id of the payoff matrix in the same order as the index of the players' ids in the list.
- **`p_payoff_matrix (np.ndarray)`** – Defined payoff matrix.

**`_setup_transfer_functions()`**

A method to setup transfer functions. This is only applicable for C\_FUNC\_TRANSFER\_FCTS. This method needs to be redefined based on the setup of the game.

**`_add_transfer_function(p_idx: int, p_transfer_fct: TransferFunction)`**

A method to add a transfer function to the GTFunction class. This method is called during the redifinition of `_setup_transfer_functions()`. This is only applicable for C\_FUNC\_TRANSFER\_FCTS.

**Parameters**

- **`p_idx (int)`** – Id of the payoff matrix in the same order as the index of the players' ids in the list.
- **`p_transfer_fct (TransferFunction)`** – Defined transfer function.

**`best_response(p_element_id: str) → float`**

A method to measure the highest possible payoff of a player/coalition in the related payoff map.

**Parameters**

**`p_element_id (str)`** – Id of a specific player/coalition.

**Returns**

The highest possible payoff of a player/coalition.

**Return type**

float

**`zero_sum() → bool`**

A method to check whether the game is a zero sum game by considering the payoff maps.

**Returns**

True means it is a zero sum game, otherwise no.

**Return type**

bool

```
class mlpro.gt.native.basics.GTPayoffMatrix(p_function: GTFunction = None, p_player_ids: list =  
None, p_logging=True)
```

Bases: [`TStamp`](#), [`Persistent`](#)

A class representing a payoff matrix for a set of players in game theory, where it includes GTFunction that has a mapping functionality.

**Parameters**

- **p\_function** ([GTFunction](#), *optional*) – Defined GTFunction for mapping functionality. The default is None.
- **p\_player\_ids** (*list*, *optional*) – List of players ids. The default is None.
- **p\_logging** – Logging functionality. The default is Log.C\_LOG\_ALL.

**C\_TYPE = 'GT Payoff Matrix'**

**get\_payoff**(*p\_strategies*: [GTStrategy](#), *p\_element\_id*: *str*) → float

A method to get the payoff for a player/coalition with respect to the selected strategies.

**Parameters**

- **p\_strategies** ([GTStrategy](#)) – Selected strategies by all players/coalitions.
- **p\_element\_id** (*str*) – ID of a specific player/coalition.

**Returns**

Payoff value.

**Return type**

float

**call\_mapping**(*p\_input*: *str*, *p\_strategies*: [GTStrategy](#)) → float

A method to run the mapping from the payoff matrix.

**Parameters**

- **p\_input** (*str*) – inputs of the payoff matrix.
- **p\_strategies** ([GTStrategy](#)) – Selected strategies by all players/coalitions.

**Returns**

Payoff of the player/coalition.

**Return type**

float

**\_call\_mapping**(*p\_input*: *str*, *p\_strategies*: [GTStrategy](#)) → float

If the payoff matrix does not use the standardized matrix by MLPro-GT. This method can be used by redefining it.

**Parameters**

- **p\_input** (*str*) – inputs of the payoff matrix.
- **p\_strategies** ([GTStrategy](#)) – Selected strategies by all players/coalitions.

**Returns**

Payoff of the player/coalition.

**Return type**

float

**best\_response\_value**(*p\_strategies*: [GTStrategy](#), *p\_element\_id*: *str*) → float

A method to calculate the gap between the payoff of the taken strategy to the best response value.

**Parameters**

- **p\_strategies** ([GTStrategy](#)) – Selected strategies by all players/coalitions.
- **p\_element\_id** (*str*) – Id of a specific player/coalition.

**Returns**

Current payoff - payoff from best response.

**Return type**

float

**\_call\_best\_response**(*p\_element\_id*: str) → float

If the payoff matrix does not use the standardized matrix by MLPro-GT. This method can be used to get the best response value by redefining it.

**Parameters**

**p\_element\_id** (str) – Id of a specific player/coalition.

**Returns**

The highest possible payoff of a player/coalition.

**Return type**

float

**zero\_sum**() → bool

A method to check whether the game is a zero sum game by considering the payoff maps.

**Returns**

True means it is a zero sum game, otherwise no.

**Return type**

bool

**\_call\_zero\_sum**() → bool

If the payoff matrix does not use the standardized matrix by MLPro-GT. This method can be used to get the detect zero sum games by redefining it.

**Returns**

True means it is a zero sum game, otherwise no.

**Return type**

bool

**class** mlpro.gt.native.basics.**GTSolver**(*p\_strategy\_space*: [MSpace](#), *p\_id*=None, *p\_visualize*: bool = False, *p\_logging*=True, *\*\*p\_param*)

Bases: [Task](#), [ScientificObject](#)

A class representing a solver (policy) in game theory.

**Parameters**

- **p\_strategy\_space** ([MSpace](#)) – Strategy space of (first) player to be added. Default = None.
- **p\_id** – Id of a player. The default is None.
- **p\_visualize** (bool, optional) – Allowing visualization. The default is False.
- **p\_logging** – Logging setup. The default is Log.C\_LOG\_ALL.
- **\*\*p\_param** – additional parameters related to the policy.

**C\_TYPE** = 'GT Solver'

**C\_NAME** = '????'

**C\_SCIREF\_TYPE** = None



**`_init_hyperparam(**p_param)`**

Implementation specific hyperparameters can be added here. Please follow these steps: a) Add each hyperparameter as an object of type `HyperParam` to the internal hyperparameter

space object `self._hyperparam_space`

- b) Create hyperparameter tuple and bind to `self._hyperparam_tuple`
- c) Set default value for each hyperparameter

**Parameters**

**p\_param** (*Dict*) – Further model specific hyperparameters, that are passed through constructor.

**`_setup_solver()`**

A method to setup a solver. This needs to be redefined based on each policy, but remains optional.

**`get_hyperparam()`** → *HyperParamTuple*

Returns the internal hyperparameter tuple to get access to single values.

**`get_strategy_space()`** → *MSpace*

A method to get the strategy space of a solver.

**Returns**

Strategy space.

**Return type**

*MSpace*

**`set_random_seed(p_seed=None)`**

Resets the internal random generator using the given seed.

**Parameters**

**p\_seed** – Seeding.

**`compute_strategy(p_payoff: GTPayoffMatrix)`** → *GTStrategy*

A method to compute a strategy from the solver.

**Parameters**

**p\_payoff** (*GTPayoffMatrix*) – Payoff matrix of a specific player.

**Returns**

The computed strategy.

**Return type**

*GTStrategy*

**`_compute_strategy(p_payoff: GTPayoffMatrix)`** → *GTStrategy*

A method to compute a strategy from the solver. This method needs to be redefined.

**Parameters**

**p\_payoff** (*GTPayoffMatrix*) – Payoff matrix of a specific player.

**Returns**

The computed strategy.

**Return type**

*GTStrategy*

```
class mlpro.gt.native.basics.GTPlayer(p_solver: list | GTSolver, p_name="", p_visualize: bool = False,
                                     p_logging=True, p_random_solver: bool = False, **p_param)
```

Bases: [GTSolver](#)

A class representing a player in game theory with at least one specific defined solver.

#### Parameters

- **p\_solver** (*Union[list, GTSolver]*) – A list of solvers or a solver.
- **p\_name** – Name of the player. The default is ‘’.
- **p\_visualize** (*bool, optional*) – Allowing visualization. The default is False.
- **p\_logging** – Logging setup. The default is Log.C\_LOG\_ALL.
- **p\_random\_solver** (*bool, optional*) – Allowing random solver. The default is False.
- **\*\*p\_param** – Additional parameters for the player.

**C\_TYPE** = 'GT Player'

**C\_NAME** = ''

**\_init\_hyperparam**(*\*\*p\_param*)

A method to initiate the related hyperparameters.

#### Parameters

**\*\*p\_param** – Additional parameters for the player.

**switch\_logging**(*p\_logging*)

A method to switch logging setup

#### Parameters

**p\_logging** – Logging setup.

**set\_log\_level**(*p\_level*)

A method to set the logging level

#### Parameters

**p\_level** – Logging level.

**get\_strategy\_space**() → [MSpace](#)

A method to get the strategy space of a player.

#### Returns

Strategy space.

#### Return type

[MSpace](#)

**set\_random\_seed**(*p\_seed=None*)

Resets the internal random generator using the given seed.

#### Parameters

**p\_seed** – Seeding.

**compute\_strategy**(*p\_payoff: GTPayoffMatrix*) → [GTStrategy](#)

A method to compute a strategy from the solver.

#### Parameters

**p\_payoff** ([GTPayoffMatrix](#)) – Payoff matrix of a specific player.

**Returns**

The computed strategy.

**Return type**

*GTStrategy*

**get\_solver()** → *GTSolver*

A method to get the solver of the player.

**Returns**

Solver.

**Return type**

*GTSolver*

**switch\_solver()**

A method to switch the solver, if the player has multiple solvers.

**class** mlpro.gt.native.basics.**GTCoalition**(*p\_name: str = ''*, *p\_coalition\_type=None*, *p\_logging=True*)

Bases: *GTPlayer*

A class representing a colation in game theory that contains a set of players or at least one player.

**Parameters**

- **p\_name** (*str*, *optional*) – Name of a coalition. The default is “”.
- **p\_coalition\_type** – Type of coalitions. The default is None.
- **p\_logging** – Logging setup. The default is Log.C\_LOG\_ALL.

**C\_TYPE** = 'GT Coalition'

**C\_NAME** = ''

**C\_COALITION\_STRATEGY** = None

**C\_COALITION\_MEAN** = 0

**C\_COALITION\_SUM** = 1

**C\_COALITION\_MIN** = 2

**C\_COALITION\_MAX** = 3

**C\_COALITION\_MEDIAN** = 4

**C\_COALITION\_MODE** = 5

**C\_COALITION\_CUSTOM** = 6

**switch\_logging**(*p\_logging*) → None

A metod to swith logging setup

**Parameters**

**p\_logging** – Loggin setup.

**switch\_adaptivity**(*p\_ada: bool*)

A method to switch adaptivity. In native GT, this is not necessary.

**Parameters**

**p\_ada** (*bool*) – adaptivity.

**set\_log\_level**(*p\_level*)

A method to set the logging level

**Parameters****p\_level** – Logging level.**add\_player**(*p\_player*: *GTPlayer*)

A method to add a player to the coalition.

**Parameters****p\_player** (*GTPlayer*) – A GT player.**get\_players**() → list

A method to get a list of players in the coalition.

**Returns**

List of GT Players.

**Return type**

list

**get\_players\_ids**() → list

A method to get the players' ids in the coalition.

**Returns**

List of ids.

**Return type**

list

**get\_player**(*p\_player\_id*) → *GTPlayer*

A method to get the object of a specific player.

**Parameters****p\_player\_id** – Id of a player.**Returns**

Object of the player.

**Return type***GTPlayer***get\_coalition\_strategy**() → int

A method to get the coalition strategy.

**Returns**

Coalition strategy.

**Return type**

int

**set\_random\_seed**(*p\_seed=None*)

Resets the internal random generator using the given seed.

**Parameters****p\_seed** – Seeding.**get\_strategy\_space**() → *ESpace*

A method to get the strategy space of the coalition.

**Returns**

Strategy space.

**Return type***ESpace***compute\_strategy**(*p\_payoff*: *GTPayoffMatrix*) → *GTStrategy*

A method to compute a combined strategy from the players in the coalition.

**Parameters****p\_payoff** (*GTPayoffMatrix*) – Payoff matrix of the coalition.**Returns**

The computed strategy.

**Return type***GTStrategy***\_custom\_coalition\_strategy**(*p\_payoff*: *GTPayoffMatrix*) → *GTStrategy*

A method for customizing the coalition strategy.

**Parameters****p\_payoff** (*GTPayoffMatrix*) – The payoff matrix of the coalition.**class** mlpro.gt.native.basics.**GTCompetition**(*p\_name*: *str* = '', *p\_logging*=*True*)Bases: *GTCoalition*

A class representing a competition in game theory that contains a set of coalitions. This suits for a competitive game.

**Parameters**

- **p\_name** (*str*, *optional*) – Name of the competition. The default is “”.
- **p\_logging** – Logging setup. The default is Log.C\_LOG\_ALL.

**Return type**

None.

**C\_TYPE** = 'GT Competition'**C\_NAME** = ''**switch\_logging**(*p\_logging*) → None

A method to switch logging setup

**Parameters****p\_logging** – Login setup.**set\_log\_level**(*p\_level*)

A method to set the logging level

**Parameters****p\_level** – Logging level.**add\_coalition**(*p\_coalition*: *GTCoalition*)

A method to add a coalition to the competition.

**Parameters****p\_coalition** (*GTCoalition*) – A coalition.**get\_coalitions**() → list

A method to get the list of coalitions in the competition.

**Returns**

List of coalitions.

**Return type**

list

**get\_coalitions\_ids()** → list

A method to get the list of coalitions' ids in the competition.

**Returns**

List of coalitions' ids.

**Return type**

list

**get\_coalition(*p\_coalition\_id*)** → *GTCoalition*

A method to get the object of a specific coalition.

**Parameters****p\_coalition\_id** – Coalition id.**Returns**

Coalition object.

**Return type***GTCoalition***get\_players()** → list

A method to get the list of players in the competition.

**Returns**

List of players.

**Return type**

list

**get\_players\_ids()** → list

A method to get the list of players' ids in the competition.

**Returns**

List of players' ids.

**Return type**

list

**get\_player(*p\_player\_id*)** → *GTPlayer*

A method to get the object of a player.

**Parameters****p\_player\_id** – Player id.**Returns**

Object of the player.

**Return type***GTPlayer***set\_random\_seed(*p\_seed=None*)**

Resets the internal random generator using the given seed.

**Parameters****p\_seed** – Seeding.

**compute\_strategy**(*p\_payoff*: *GTPayoffMatrix*) → *GTStrategy*

A method to compute the strategy of each coalition.

**Parameters**

**p\_payoff** (*GTPayoffMatrix*) – Payoff matrices.

**Returns**

The computed strategy.

**Return type**

*GTStrategy*

**get\_strategy\_space**() → *ESpace*

A method to get the strategy space of the competition.

**Returns**

Strategy space.

**Return type**

*ESpace*

**class** mlpro.gt.native.basics.**GTDataStoring**(*p\_space*: *Set* = *None*)

Bases: *DataStoring*

A method for data storing of the game.

**Parameters**

**p\_space** (*Set*, *optional*) – Spaces. The default is *None*.

**C\_VAR0** = 'Trial'

**C\_VAR\_CYCLE** = 'Cycle'

**C\_VAR\_DAY** = 'Day'

**C\_VAR\_SEC** = 'Second'

**C\_VAR\_MICROSEC** = 'Microsecond'

**get\_variables**()

A method to get variables.

**get\_space**()

A method to get space..

**add\_trial**(*p\_trial\_id*)

A method to add a trial id into the data storing.

**Parameters**

**p\_trial\_id** – Trial id.

**memorize\_row**(*p\_cycle\_id*, *p\_tstamp*: *timedelta*, *p\_data*)

A method to add data to the data storing

**Parameters**

- **p\_cycle\_id** – Cyle id.
- **p\_tstamp** (*timedelta*) – Time stamp.
- **p\_data** – Data to be stored.

```
class mlpro.gt.native.basics.GTGame(p_mode=0, p_ada=False, p_cycle_limit=1, p_visualize: bool =
                                   False, p_logging=True)
```

Bases: [Scenario](#)

A class representing a game in game theory.

#### Parameters

- **p\_mode** – Operation mode. See `bf.ops.Mode.C_VALID_MODES` for valid values. Default = `Mode.C_MODE_SIM`.
- **p\_ada** – Boolean switch for adaptivity. In the native GT, this is always switched off. The default is `False`.
- **p\_cycle\_limit** – Maximum number of cycles.. The default is 1.
- **p\_visualize** (*bool, optional*) – Boolean switch for visualisation. The default is `False`.
- **p\_logging** – Log level (see constants of class `Log`). The default is `Log.C_LOG_ALL`.

**C\_TYPE** = 'GT Game'

**C\_NAME** = ''

**C\_LATENCY** = `datetime.timedelta(seconds=1)`

**\_setup**(*p\_mode, p\_ada: bool, p\_visualize: bool, p\_logging*) → [Model](#)

Custom setup of GT Game. Payoff matrix has to be defined here as `self._payoff`.

#### Parameters

- **p\_mode** – Operation mode. See `Mode.C_VALID_MODES` for valid values. Default = `Mode.C_MODE_SIM`
- **p\_ada** (*bool*) – Boolean switch for adaptivity.
- **p\_visualize** (*bool*) – Boolean switch for visualisation.
- **p\_logging** – Log level (see constants of class `Log`).

#### Returns

**player** – GTPlayer model (object of type `GTPlayer`, `GTCoalition` or `GTCompetition`).

#### Return type

[GTPlayer](#)

**\_run\_cycle**()

A method to run a cycle in the defined game

#### Return type

`False, False, False, False`

**\_get\_evaluation**(*p\_coalition\_id: str, p\_coalition: [GTCoalition](#)*) → `float | list`

A method to get the evaluation of a coalition in the form of payoff matrix.

#### Parameters

- **p\_coalition\_id** (*str*) – Coalition id.
- **p\_coalition** ([GTCoalition](#)) – Coalition object.

#### Returns

Payoff of the respective coalition.



**Return type**

Union[float,list]

**connect\_data\_logger**(*p\_ds\_strategies*: GTDataStoring = None, *p\_ds\_payoffs*: GTDataStoring = None)

A method to connect with the data logger from GTDataStoring.

**Parameters**

- **p\_ds\_strategies** (GTDataStoring, optional) – Object of GTDataStoring of strategies. The default is None.
- **p\_ds\_payoffs** (GTDataStoring, optional) – Object of GTDataStoring of payoffs. The default is None.

**is\_zerogame**() → bool

A method to identify whether it is a zero-sum game.

**Returns**

True means zero-sum game, otherwise not.

**Return type**

bool

**\_is\_bestresponse**(*p\_coalition\_id*: str, *p\_coalition*: GTCoalition) → float

A method to identify whether the best response value of a coalition.

**Parameters**

- **p\_coalition\_id** (str) – Coalition id.
- **p\_coalition** (GTCoalition) – Coalition object.

**Returns**

The best response value.

**Return type**

float

**get\_latency**() → timedelta

A method to get the latency of the game

**Returns**

Latency.

**Return type**

timedelta

```
class mlpro.gt.native.basics.GTTrainingResults(p_scenario: GTGame, p_run: int, p_cycle_id: int,
                                              p_logging='W')
```

Bases: *TrainingResults*

Results of a native GT training.

**Parameters**

- **p\_scenario** (GTScenario) – Related native GT scenario.
- **p\_run** (int) – Run id.
- **p\_cycle\_id** (int) – Id of first cycle of this run.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL

**C\_NAME** = 'GT Training Results'

**C\_FNAME\_COAL\_STRATEGIES** = 'strategies'

**C\_FNAME\_COAL\_PAYOFFS** = 'payoffs'

**save**(*p\_path*, *p\_filename*='summary.csv') → bool

A method to save the training results

**Parameters**

- **p\_path** – Saving path.
- **p\_filename** – Name and format of the file. The default is 'summary.csv'.

**Returns**

True means successful, otherwise failed.

**Return type**

bool

**class** mlpro.gt.native.basics.**GTTraining**(\*\**p\_kwargs*)

Bases: *Training*

Template class for a GT training.

**Parameters**

- **p\_game\_cls** – Name of GT game class, compatible to/inherited from class GTGame.
- **p\_cycle\_limit** (*int*) – Maximum number of training cycles (0=no limit). Default = 0.
- **p\_adaptation\_limit** (*int*) – Maximum number of adaptations (0=no limit). Default = 0.
- **p\_hpt** (*HyperParamTuner*) – Optional hyperparameter tuner (see class HyperParamTuner). Default = None.
- **p\_hpt\_trials** (*int*) – Optional number of hyperparameter tuning trials. Default = 0.
- **p\_path** (*str*) – Optional destination path to store training data. Default = None.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default = Log.C\_LOG\_WE.
- **p\_collect\_strategy** – Collect data of selected strategies. Default = False.
- **p\_collect\_payoff** – Collect data of obtained payoffs. Default = False.
- **p\_init\_seed** – Seeding. Default = 0.

**C\_TYPE** = 'GT Training'

**C\_NAME** = 'Native GT Training'

**C\_CLS\_RESULTS**

alias of *GTTrainingResults*

**\_init\_results**() → *GTTrainingResults*

A method to initialise data storing functionality.

**Returns**

Object of GTTrainingResults.

**Return type**

*GTTrainingResults*

**\_init\_trial()**

A method to initialise a trial.

**\_close\_trial()**

A method to close/stop a trial.

**\_run\_cycle()** → bool

A method to run a cycle.

**Returns**

False.

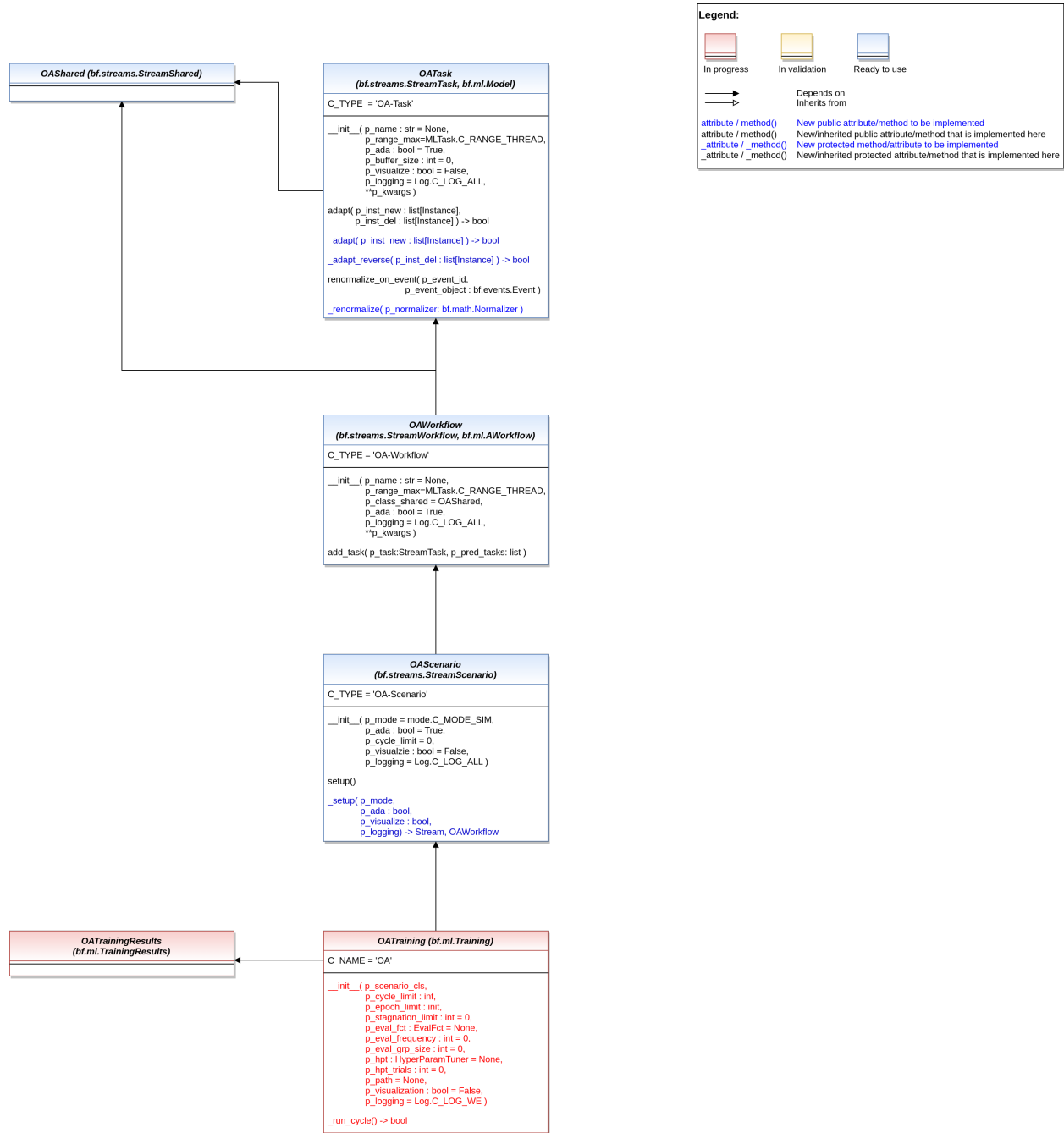
**Return type**

bool

### 11.1.5 MLPro-OA - Online Adaptivity

#### OA-STREAMS - Online-adaptive Stream Processing

## OA-STREAMS-BASICS - Basic Classes for Processing



Ver. 0.8.0 (2023-12-20)

Core classes for online adaptive stream processing.

**class** mlpro.oa.streams.basics.OAShared(*p\_range: int = 2*)Bases: *StreamShared*

Template class for shared objects in the context of online adaptive stream processing.

**class** mlpro.oa.streams.basics.OATask(*p\_name: str = None, p\_range\_max=1, p\_adapt: bool = True, p\_buffer\_size: int = 0, p\_duplicate\_data: bool = False, p\_visualize: bool = False, p\_logging=True, \*\*p\_kwargs*)

Bases: [StreamTask](#), [Model](#)

Template class for online adaptive ML tasks.

#### Parameters

- **p\_name** (*str*) – Optional name of the task. Default is None.
- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class [Range](#). Default is [Range.C\\_RANGE\\_PROCESS](#).
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_buffer\_size** (*int*) – Initial size of internal data buffer. Default = 0 (no buffering).
- **p\_duplicate\_data** (*bool*) – If True, instances will be duplicated before processing. Default = False.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class [Log](#)). Default: [Log.C\\_LOG\\_ALL](#)
- **p\_kwargs** (*dict*) – Further optional named parameters.

**C\_TYPE** = 'OA-Task'

**C\_PLOT\_ACTIVE**: *bool* = True

**C\_PLOT\_STANDALONE**: *bool* = True

**C\_PLOT\_VALID\_VIEWS**: *list* = ['2D', '3D', 'ND']

**C\_PLOT\_DEFAULT\_VIEW**: *str* = 'ND'

**adapt**(*p\_inst\_new*: *List*[[Instance](#)], *p\_inst\_del*: *List*[[Instance](#)]) → *bool*

Adapts the model by calling the custom method `_adapt()`.

#### Parameters

**p\_kwargs** (*dict*) – All parameters that are needed for the adaption. Depends on the specific higher context.

#### Returns

**adapted** – True, if something has been adapted. False otherwise.

#### Return type

*bool*

**\_adapt**(*p\_inst\_new*: *List*[[Instance](#)]) → *bool*

Obligatory custom method for adaptations during regular operation.

#### Parameters

**p\_inst\_new** (*list*) – List of new stream instances to be processed.

#### Returns

**adapted** – True, if something has been adapted. False otherwise.

#### Return type

*bool*

**\_adapt\_reverse**(*p\_inst\_del*: *List*[[Instance](#)]) → *bool*

Optional custom method for reverse adaptations during regular operation.

#### Parameters

**p\_inst\_del** (*list*) – List of obsolete stream instances to be removed.

**Returns**

**adapted** – True, if something has been adapted. False otherwise.

**Return type**

bool

**\_renormalize**(*p\_normalizer*: [Normalizer](#))

Custom method to renormalize internally buffered data using the given normalizer object. This is necessary after an adaptation of a related predecessor normalizer task. See method `renormalize_on_event()` for further details.

**Parameters**

**p\_normalizer** ([Normalizer](#)) – Normalizer object to be applied on task-specific

**renormalize\_on\_event**(*p\_event\_id*: str, *p\_event\_object*: [Event](#))

Event handler method to be registered on event `Model.C_EVENT_ADAPTED` of an online adaptive normalizer task. It carries out the task-specific renormalization of internally buffered data by calling the custom method `_renormalize()`.

**Parameters**

- **p\_event\_id** (str) – Unique event id
- **p\_event\_object** ([Event](#)) – Event object with further context informations

**\_so**: [Shared](#)

**\_range**: int

**\_plot\_settings**: [PlotSettings](#)

```
class mlpro.oa.streams.basics.OAWorkflow(p_name: str = None, p_range_max=1, p_class_shared=<class  
    'mlpro.oa.streams.basics.OAShared'>, p_ada: bool = True,  
    p_visualize: bool = False, p_logging=True, **p_kwargs)
```

Bases: [StreamWorkflow](#), [AWorkflow](#)

Online adaptive workflow based on a stream-workflow and an adaptive workflow.

**Parameters**

- **p\_name** (str) – Optional name of the workflow. Default is None.
- **p\_range\_max** (int) – Maximum range of asynchronicity. See class `Range`. Default is `Range.C_RANGE_PROCESS`.
- **p\_class\_shared** – Optional class for a shared object (class `OAShared` or a child class of `OAShared`)
- **p\_ada** (bool) – Boolean switch for adaptivity. Default = True.
- **p\_visualize** (bool) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class `Log`). Default: `Log.C_LOG_ALL`
- **p\_kwargs** (dict) – Further optional named parameters.

**C\_TYPE** = 'OA-Workflow'

**add\_task**(*p\_task*: [StreamTask](#), *p\_pred\_tasks*: list = None)

Adds a task to the workflow.

**Parameters**

- **p\_task** ([Task](#)) – Task object to be added.

- **p\_pred\_tasks** (*list*) – Optional list of predecessor task objects

**\_so:** *Shared*

**\_range:** *int*

**\_plot\_settings:** *PlotSettings*

**class** mlpro.oa.streams.basics.**OAScenario**(*p\_mode=0, p\_ada: bool = True, p\_cycle\_limit=0, p\_visualize: bool = False, p\_logging=True*)

Bases: *StreamScenario*

Template class for stream based scenarios with online adaptive workflows.

#### Parameters

- **p\_mode** – Operation mode. See `bf.ops.Mode.C_VALID_MODES` for valid values. Default = `Mode.C_MODE_SIM`.
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = `True`.
- **p\_cycle\_limit** (*int*) – Maximum number of cycles (0=no limit, -1=get from env). Default = 0.
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = `False`.
- **p\_logging** – Log level (see constants of class `mlpro.bf.various.Log`). Default = `Log.C_LOG_WE`.

**C\_TYPE** = 'OA-Scenario'

#### **setup()**

Specialized method to set up an oa stream scenario. It is automatically called by the constructor and calls in turn the custom method `_setup()`.

**\_setup**(*p\_mode, p\_ada: bool, p\_visualize: bool, p\_logging*)

Custom method to set up a stream scenario consisting of a stream and a processing stream workflow.

#### Parameters

- **p\_mode** – Operation mode. See `Mode.C_VALID_MODES` for valid values. Default = `Mode.C_MODE_SIM`.
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = `True`.
- **p\_visualize** (*bool*) – Boolean switch for visualisation.
- **p\_logging** – Log level (see constants of class `Log`). Default: `Log.C_LOG_ALL`.

#### Returns

- **stream** (*Stream*) – A stream object.
- **workflow** (*OAWorkflow*) – An online adaptive stream workflow object.

**\_stream:** *Stream*

**\_iterator:** *Stream*

**\_workflow:** *StreamWorkflow*

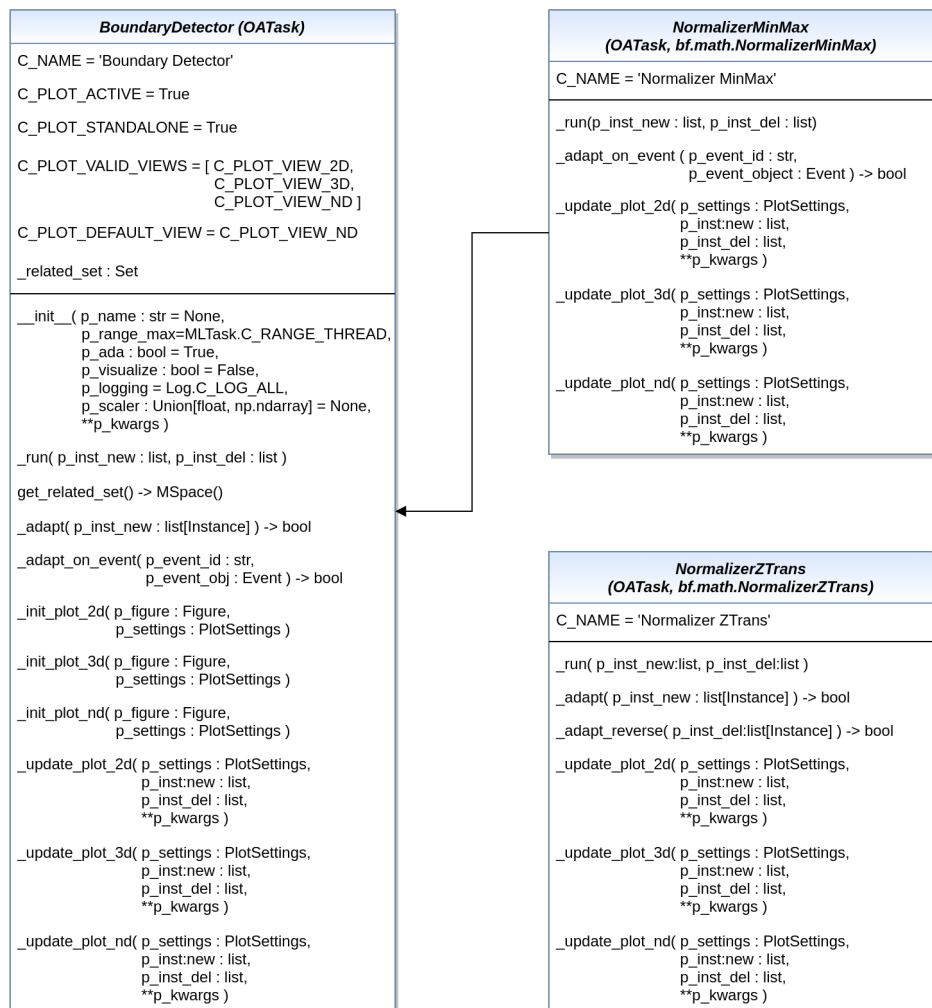
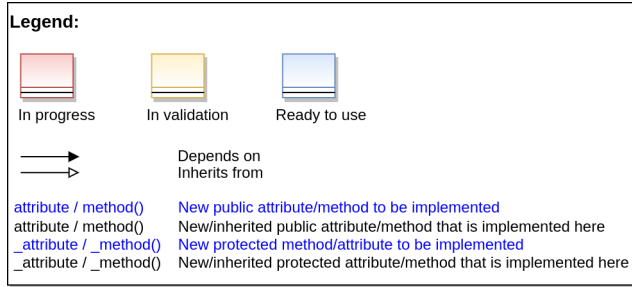
**\_plot\_settings:** *PlotSettings*

```
class mlpro.oa.streams.basics.OATrainingResults(p_scenario: Scenario, p_run, p_cycle_id,
                                                p_logging='W')
    Bases: TrainingResults
    ...
class mlpro.oa.streams.basics.OATraining(**p_kwargs)
    Bases: Training
    ...
    C_NAME = 'OA'
```

## OA-STREAMS-TASKS - Online-adaptive Stream Tasks



## OA-STREAMS-TASKS - Data Preprocessing



## Boundary Detector

Ver. 1.2.4 (2023-11-19)

This module provides pool of boundary detector object further used in the context of online adaptivity.

```
class mlpro.oa.streams.tasks.boundarydetectors.BoundaryDetector(p_name: str = None,  
                                                                p_range_max=1, p_adapt: bool =  
                                                                True, p_duplicate_data: bool =  
                                                                False, p_visualize: bool = False,  
                                                                p_logging=True, p_scaler: float  
                                                                | Iterable = array([1.]),  
                                                                **p_kwargs)
```

Bases: [OATask](#)

This is the base class for Boundary Detector object. It raises event when a change in the current boundaries is detected based on the new data instances

### Parameters

- **p\_name** (*str*, *Optional*.) – Name of the task.
- **p\_range\_max** – Processing range of the task. Default is thread.
- **p\_adapt** (*bool*) – True if the task has adaptivity. Default is True.
- **p\_duplicate\_data** (*bool*) – If True, instances will be duplicated before processing. Default = False.
- **p\_visualize** (*bool*) – True to turn on the visualization.
- **p\_logging** – Logging level for the task, default is Log all.
- **p\_scaler** (*float*, *np.ndarray*) – A scaler vector to scale the detected boundaries.

**C\_NAME** = 'Boundary Detector'

**C\_PLOT\_ND\_XLABEL\_FEATURE** = 'Features'

**C\_PLOT\_ND\_YLABEL** = 'Boundaries'

**C\_PLOT\_VALID\_VIEWS: list** = ['ND']

**\_adapt**(*p\_inst\_new: List[Instance]*)

Method to check if the new instances exceed the current boundaries of the Set.

### Parameters

**p\_inst\_new** (*list*) – List of new instance/s added to the workflow

### Returns

**adapted** – Returns true if there is a change of boundaries, false otherwise.

### Return type

bool

**\_run**(*p\_inst\_new: List[Element]*, *p\_inst\_del: List[Element]*)

Method to run the boundary detector task

### Parameters

- **p\_inst\_new** (*list*) – List of new instance/s added to the workflow
- **p\_inst\_del** (*list*) – List of old obsolete instance/s removed from the workflow

**`_adapt_on_event`**(*p\_event\_id*: str, *p\_event\_object*: [Event](#))

Event handler for Boundary Detector that adapts if the related event is raised.

**Parameters**

- **`p_event_id`** – The event id related to the adaptation.
- **`p_event_obj`** ([Event](#)) – The event object related to the raised event.

**Returns**

Returns true if adapted, false otherwise.

**Return type**

bool

**`get_related_set`**()

**`_init_plot_nd`**(*p\_figure*: [Figure](#), *p\_settings*: [PlotSettings](#))

Custom method to initialize plot for Boundary Detectors tasks for N-dimensional plotting.

**Parameters**

- **`p_figure`** ([Figure](#)) – Figure to host the plot
- **`p_settings`** ([PlotSettings](#)) – PlotSettings objects with specific settings for the plot

**`_update_plot_nd`**(*p\_settings*: [PlotSettings](#), *p\_inst\_new*: list, *p\_inst\_del*: list, *\*\*p\_kwargs*)

Default N-dimensional plotting implementation for Boundary Detector tasks. See class `mlpro.bf.plot.Plottable` for more details.

**Parameters**

- **`p_settings`** ([PlotSettings](#)) – Object with further plot settings.
- **`p_inst_new`** (list) – List of new stream instances to be plotted.
- **`p_inst_del`** (list) – List of obsolete stream instances to be removed.
- **`p_kwargs`** (dict) – Further optional plot parameters.

## Normalizers

Ver. 1.2.2 (2023-05-22)

This module provides implementation for adaptive normalizers for MinMax Normalization and ZTransformation

```
class mlpro.oa.streams.tasks.normalizers.NormalizerMinMax(p_name: str = None, p_range_max=1,
                                                         p_ada: bool = True, p_duplicate_data:
                                                         bool = False, p_visualize: bool = False,
                                                         p_logging=True, **p_kwargs)
```

Bases: [OATask](#), [NormalizerMinMax](#)

Class with functionality for adaptive normalization of instances using MinMax Normalization.

**Parameters**

- **`p_name`** (str, optional) – Name of the task.
- **`p_range_max`** – Processing range of the task, default is a Thread.
- **`p_ada`** – True if the task has adaptivity, default is true.
- **`p_duplicate_data`** (bool) – If True, instances will be duplicated before processing. Default = False.

- **p\_visualize** – True for visualization, false by default.
- **p\_logging** – Logging level of the task. Default is Log.C\_LOG\_ALL
- **p\_kwargs** – Additional task parameters

**C\_NAME** = 'Normalizer MinMax'

**\_run**(*p\_inst\_new*: list, *p\_inst\_del*: list)

Custom method to for run MinMax Normalizer task for normalizing new instances and denormalizing deleted instances.

#### Parameters

- **p\_inst\_new** (*list*) – List of new instances in the workflow
- **p\_inst\_del** (*list*) – List of deleted instances in the workflow

**\_adapt\_on\_event**(*p\_event\_id*: str, *p\_event\_object*: [Event](#)) → bool

Custom method to adapt the MinMax normalizer parameters based on event raised by Boundary object for changed boundaries.

#### Parameters

- **p\_event\_id** (*str*) – Event id of the raised event
- **p\_event\_obj** ([Event](#)) – Event object that raises the corresponding event

#### Returns

**adapted** – Returns True, if the task has adapted. False otherwise.

#### Return type

bool

**\_update\_plot\_2d**(*p\_settings*: [PlotSettings](#), *p\_inst\_new*: list, *p\_inst\_del*: list, **\*\*p\_kwargs**)

Method to update the 2d plot for Normalizer. Extended to renormalize the obsolete data on change of parameters.

#### Parameters

- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.
- **p\_inst\_new** (*list*) – List of new stream instances to be plotted.
- **p\_inst\_del** (*list*) – List of obsolete stream instances to be removed.
- **p\_kwargs** (*dict*) – Further optional plot parameters.

**\_update\_plot\_3d**(*p\_settings*: [PlotSettings](#), *p\_inst\_new*: list, *p\_inst\_del*: list, **\*\*p\_kwargs**)

Method to update the 3d plot for Normalizer. Extended to renormalize the obsolete data on change of parameters.

#### Parameters

- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.
- **p\_inst\_new** (*list*) – List of new stream instances to be plotted.
- **p\_inst\_del** (*list*) – List of obsolete stream instances to be removed.
- **p\_kwargs** (*dict*) – Further optional plot parameters.

**\_update\_plot\_nd**(*p\_settings*: [PlotSettings](#), *p\_inst\_new*: list, *p\_inst\_del*: list, **\*\*p\_kwargs**)

Method to update the nd plot for Normalizer. Extended to renormalize the obsolete data on change of parameters.

**Parameters**

- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.
- **p\_inst\_new** (*list*) – List of new stream instances to be plotted.
- **p\_inst\_del** (*list*) – List of obsolete stream instances to be removed.
- **p\_kwargs** (*dict*) – Further optional plot parameters.

```
class mlpro.oa.streams.tasks.normalizers.NormalizerZTransform(p_name: str = None,
                                                             p_range_max=1, p_ada: bool =
                                                             True, p_duplicate_data: bool =
                                                             False, p_visualize=False,
                                                             p_logging=True, **p_kwargs)
```

Bases: [OATask](#), [NormalizerZTrans](#)

Class with functionality of adaptive normalization of instances with Z-Transformation

**Parameters**

- **p\_name** (*str, optional*) – Name of the task.
- **p\_range\_max** – Processing range of the task, default is a Thread.
- **p\_ada** – True if the task has adaptivity, default is true.
- **p\_duplicate\_data** (*bool*) – If True, instances will be duplicated before processing. Default = False.
- **p\_visualize** – True for visualization, false by default.
- **p\_logging** – Logging level of the task. Default is Log.C\_LOG\_ALL
- **p\_kwargs** – Additional task parameters

**C\_NAME = 'Normalizer Z Transform'**

**\_run**(*p\_inst\_new: list, p\_inst\_del: list*)

Custom method to for run Z-transform task for normalizing new instances and denormalizing deleted instances.

**Parameters**

- **p\_inst\_new** (*list*) – List of new instances in the workflow
- **p\_inst\_del** (*list*) – List of deleted instances in the workflow

**\_adapt**(*p\_inst\_new: List[Instance]*) → bool

Custom method to for adapting of Z-transform parameters on new instances.

**Parameters**

- **p\_inst\_new** (*list*) – List of new instances in the workflow

**Returns**

**adapted** – Returns True, if task has adapted.

**Return type**

bool

**\_adapt\_reverse**(*p\_inst\_del: List[Instance]*) → bool

Custom method to for adapting of Z-transform parameters on deleted instances.

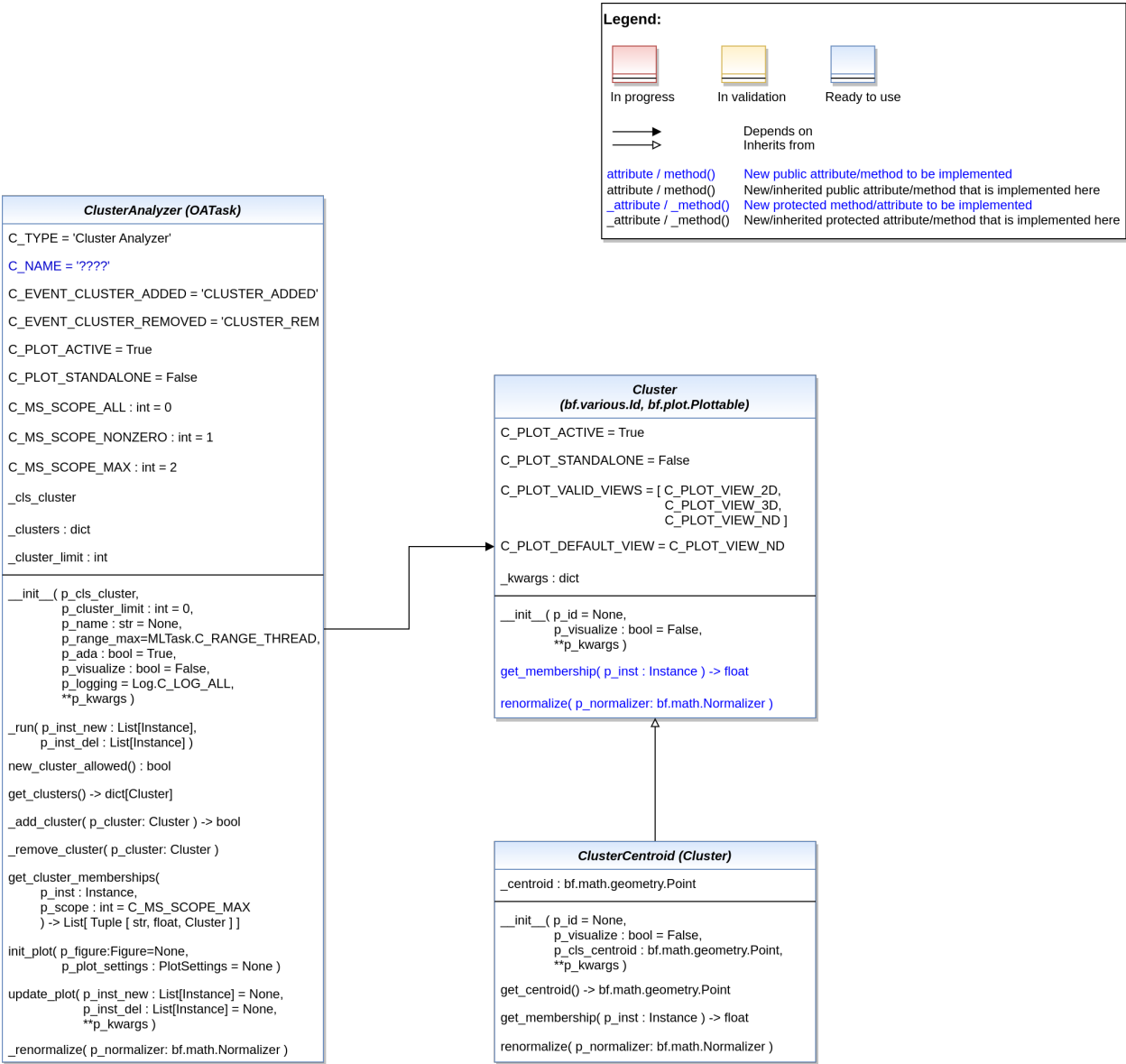
**Parameters**

- **p\_inst\_del** (*list*) – List of deleted instances in the workflow

**Returns**  
**adapted** – Returns True, if task has adapted.

**Return type**  
bool

OA-STREAMS-TASKS - Cluster Analysis



## OA-STREAMS-TASKS - Anomaly Detectors

content/99\_appendices/appendix2/sub/core/mlpro\_oa/streams/tasks/images/MLPro-OA-Anomaly\_Detectors\_class

## OA-STREAMS - Online-adaptive Stream Processing

content/99\_appendices/appendix2/sub/core/mlpro\_oa/images/01\_environments/MLPro-OA-Proc\_class\_diagram.dr

Ver. 0.8.0 (2023-12-20)

Core classes for online adaptive stream processing.

**class** mlpro.oa.streams.basics.OAShared(*p\_range: int = 2*)

Bases: *StreamShared*

Template class for shared objects in the context of online adaptive stream processing.

**class** mlpro.oa.streams.basics.OATask(*p\_name: str = None, p\_range\_max=1, p\_ada: bool = True, p\_buffer\_size: int = 0, p\_duplicate\_data: bool = False, p\_visualize: bool = False, p\_logging=True, \*\*p\_kwargs*)

Bases: *StreamTask, Model*

Template class for online adaptive ML tasks.

### Parameters

- **p\_name** (*str*) – Optional name of the task. Default is None.
- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class Range. Default is Range.C\_RANGE\_PROCESS.
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_buffer\_size** (*int*) – Initial size of internal data buffer. Default = 0 (no buffering).
- **p\_duplicate\_data** (*bool*) – If True, instances will be duplicated before processing. Default = False.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL
- **p\_kwargs** (*dict*) – Further optional named parameters.

**C\_TYPE** = 'OA-Task'

**C\_PLOT\_ACTIVE**: *bool* = True

**C\_PLOT\_STANDALONE**: *bool* = True

**C\_PLOT\_VALID\_VIEWS**: *list* = ['2D', '3D', 'ND']

**C\_PLOT\_DEFAULT\_VIEW**: *str* = 'ND'

**adapt**(*p\_inst\_new: List[Instance], p\_inst\_del: List[Instance]*) → *bool*

Adapts the model by calling the custom method `_adapt()`.

**Parameters**

**p\_kwargs** (*dict*) – All parameters that are needed for the adaption. Depends on the specific higher context.

**Returns**

**adapted** – True, if something has been adapted. False otherwise.

**Return type**

bool

**\_adapt**(*p\_inst\_new*: *List*[*Instance*]) → bool

Obligatory custom method for adaptations during regular operation.

**Parameters**

**p\_inst\_new** (*list*) – List of new stream instances to be processed.

**Returns**

**adapted** – True, if something has been adapted. False otherwise.

**Return type**

bool

**\_adapt\_reverse**(*p\_inst\_del*: *List*[*Instance*]) → bool

Optional custom method for reverse adaptations during regular operation.

**Parameters**

**p\_inst\_del** (*list*) – List of obsolete stream instances to be removed.

**Returns**

**adapted** – True, if something has been adapted. False otherwise.

**Return type**

bool

**\_renormalize**(*p\_normalizer*: *Normalizer*)

Custom method to renormalize internally buffered data using the given normalizer object. This is necessary after an adaptation of a related predecessor normalizer task. See method `renormalize_on_event()` for further details.

**Parameters**

**p\_normalizer** (*Normalizer*) – Normalizer object to be applied on task-specific

**renormalize\_on\_event**(*p\_event\_id*: *str*, *p\_event\_object*: *Event*)

Event handler method to be registered on event `Model.C_EVENT_ADAPTED` of an online adaptive normalizer task. It carries out the task-specific renormalization of internally buffered data by calling the custom method `_renormalize()`.

**Parameters**

- **p\_event\_id** (*str*) – Unique event id
- **p\_event\_object** (*Event*) – Event object with further context informations

```
class mlpro.oa.streams.basics.OAWorkflow(p_name: str = None, p_range_max=1, p_class_shared=<class  
'mlpro.oa.streams.basics.OAShared'>, p_ada: bool = True,  
p_visualize: bool = False, p_logging=True, **p_kwargs)
```

Bases: *StreamWorkflow*, *AWorkflow*

Online adaptive workflow based on a stream-workflow and an adaptive workflow.

**Parameters**

- **p\_name** (*str*) – Optional name of the workflow. Default is None.



- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class Range. Default is Range.C\_RANGE\_PROCESS.
- **p\_class\_shared** – Optional class for a shared object (class OAShared or a child class of OAShared)
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL
- **p\_kwargs** (*dict*) – Further optional named parameters.

**C\_TYPE** = 'OA-Workflow'

**add\_task**(*p\_task*: StreamTask, *p\_pred\_tasks*: list = None)

Adds a task to the workflow.

#### Parameters

- **p\_task** (Task) – Task object to be added.
- **p\_pred\_tasks** (list) – Optional list of predecessor task objects

**class** mlpro.oa.streams.basics.OAScenario(*p\_mode*=0, *p\_ada*: bool = True, *p\_cycle\_limit*=0, *p\_visualize*: bool = False, *p\_logging*=True)

Bases: StreamScenario

Template class for stream based scenarios with online adaptive workflows.

#### Parameters

- **p\_mode** – Operation mode. See bf.ops.Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM.
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_cycle\_limit** (*int*) – Maximum number of cycles (0=no limit, -1=get from env). Default = 0.
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p\_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C\_LOG\_WE.

**C\_TYPE** = 'OA-Scenario'

**setup()**

Specialized method to set up an oa stream scenario. It is automatically called by the constructor and calls in turn the custom method \_setup().

**\_setup**(*p\_mode*, *p\_ada*: bool, *p\_visualize*: bool, *p\_logging*)

Custom method to set up a stream scenario consisting of a stream and a processing stream workflow.

#### Parameters

- **p\_mode** – Operation mode. See Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM.
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_visualize** (*bool*) – Boolean switch for visualisation.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL.

**Returns**

- **stream** (*Stream*) – A stream object.
- **workflow** (*OAWorkflow*) – An online adaptive stream workflow object.

```
class mlpro.oa.streams.basics.OATrainingResults(p_scenario: Scenario, p_run, p_cycle_id,
                                              p_logging='W')
```

Bases: *TrainingResults*

...

```
class mlpro.oa.streams.basics.OATraining(**p_kwargs)
```

Bases: *Training*

...

**C\_NAME** = 'OA'

**OA-SYSTEMS - Online-adaptive Systems**

## 11.2 Pool Objects

### 11.2.1 MLPro-BF - Basic Functions

**BF-STREAMS - Stream Processing****Native Sample Streams****Data from CSV files**

Ver. 1.1.2 (2023-04-17)

This module provides the native stream class `StreamMLProCSV`. This stream provides a functionality to convert csv file to a MLPro compatible stream data.

```
class mlpro.bf.streams.streams.csv_file.StreamMLProCSV(p_id=None, p_name: str = "",
                                                       p_num_instances: int = 0, p_version: str = "",
                                                       p_feature_space: MSpace = None,
                                                       p_label_space: MSpace = None, p_sampler:
                                                       Sampler = None, p_mode=0,
                                                       p_logging=True, **p_kwargs)
```

Bases: *Stream*

Reusable class for converting data from csv files to MLPro's data streams.

**Parameters**

- **p\_id** – Optional id of the stream. Default = None.
- **p\_name** (*str*) – Optional name of the stream. Default = "".
- **p\_num\_instances** (*int*) – Optional number of instances in the stream. Default = 0.
- **p\_version** (*str*) – Optional version of the stream. Default = "".
- **p\_feature\_space** (*MSpace*) – Optional feature space. Default = None.
- **p\_label\_space** (*MSpace*) – Optional label space. Default = None.

- **p\_sampler** – Optional sampler. Default: None.
- **p\_mode** – Operation mode. Default: Mode.C\_MODE\_SIM.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL.
- **p\_path\_load** – Path of the loaded file.
- **p\_csv\_filename** – File name of the loaded CSV file.
- **p\_delimiter** – Delimiter of the CSV data. Default: “ “
- **p\_frame** – Availability of framed in the loaded CSV file. Default: True
- **p\_header** – Availability of header in the first row of the loaded CSV file. Default: True
- **p\_list\_features** – List of the file’s headers that is loaded as features in the stream. Default: None
- **p\_list\_labels** – List of the file’s headers that is loaded as labels in the stream. Default: None

**C\_ID** = 'CSV2MLPro'

**C\_TYPE** = 'Stream CSV File'

**C\_NAME** = ''

**C\_VERSION** = '1.0.0'

**C\_SCIREF\_TYPE** = 'Online'

**C\_SCIREF\_AUTHOR** = 'MLPro'

**C\_SCIREF\_URL** = 'https://mlpro.readthedocs.io'

**set\_options(\*\*p\_kwargs)**

Method to set specific options for the stream. The possible options depend on the stream provider and stream itself.

**\_setup\_feature\_space()** → *MSpace*

Custom method to set up the feature space of the stream. It is called by method `get_feature_space()`.

**Returns**

**feature\_space** – Feature space of the stream.

**Return type**

*MSpace*

**\_setup\_label\_space()** → *MSpace*

Custom method to set up the label space of the stream. It is called by method `get_label_space()`.

**Returns**

**label\_space** – Label space of the stream.

**Return type**

*MSpace*

**\_init\_dataset()**

**\_reset()**

Custom reset method for data stream. See method `__iter__()` for more details.

**\_get\_next()** → *Instance*

Custom method to determine the next data stream instance. At the end of the stream exception `StopIteration` is to be raised. See method `__next__()` for more details.

**Returns**

**instance** – Next instance of data stream or None.

**Return type**

*Instance*

## 10D Random Instances

Ver. 1.0.0 (2022-12-13)

This module provides the native stream class `StreamMLProRnd10D`. This stream provides 1000 instances with 10-dimensional random feature data and 2-dimensional random label data.

**class** `mlpro.bf.streams.streams.rnd10d.StreamMLProRnd10D`(*p\_logging=True, \*\*p\_kwargs*)

Bases: `StreamMLProBase`

Demo stream consisting of 1000 instances with 10-dimensional random feature data and 2-dimensional label data. All values are in range defined by attribute `C_BOUNDARIES`.

**C\_NUM\_INSTANCES** = 1000

Number of instances.

**C\_BOUNDARIES** = [-10, 10]

Boundaries for all random values.

**C\_ID** = 'Rnd10Dx1000'

**C\_NAME** = 'Random 10D x 1000'

**C\_VERSION** = '1.0.0'

**C\_NUM\_INSTANCES** = 1000

**C\_SCIREF\_ABSTRACT** = 'Demo stream of 1000 instances with 10-dimensional random feature data and 2-dimensional label data.'

**C\_BOUNDARIES** = [-10, 10]

**\_setup\_feature\_space()** → *MSpace*

Custom method to set up the feature space of the stream. It is called by method `get_feature_space()`.

**Returns**

**feature\_space** – Feature space of the stream.

**Return type**

*MSpace*

**\_setup\_label\_space()** → *MSpace*

Custom method to set up the label space of the stream. It is called by method `get_label_space()`.

**Returns**

**label\_space** – Label space of the stream.

**Return type**

*MSpace*

**`_init_dataset()`**

Custom method to generate stream data as a numpy array named `self._dataset`.

**`set_random_seed(p_seed=None)`**

Resets the internal random generator using the given seed.

**`_get_next()`** → *Instance*

Custom method to determine the next data stream instance. At the end of the stream exception `StopIteration` is to be raised. See method `__next__()` for more details.

**Returns**

**instance** – Next instance of data stream or None.

**Return type**

*Instance*

**2D Double Spiral**

Ver. 1.0.0 (2022-12-14)

This module provides the native stream class `DoubleSpiral2D`. It provides 721 instances with 2-dimensional feature data that follow a double spiral pattern.

```
class mlpro.bf.streams.streams.doublespiral2d.DoubleSpiral2D(p_logging=True, **p_kwargs)
```

Bases: `StreamMLProBase`

**C\_ID** = 'DoubleSpiral2D'

**C\_NAME** = 'Double Spiral 2D x 721'

**C\_VERSION** = '1.0.0'

**C\_NUM\_INSTANCES** = 721

**C\_SCIREF\_ABSTRACT** = 'This benchmark test generates 721 2-dimensional inputs positioned in a double spiral.'

**C\_BOUNDARIES** = [-10, 10]

**`_setup_feature_space()`** → *MSpace*

Custom method to set up the feature space of the stream. It is called by method `get_feature_space()`.

**Returns**

**feature\_space** – Feature space of the stream.

**Return type**

*MSpace*

**`_init_dataset()`**

Custom method to generate stream data as a numpy array named `self._dataset`.

## Multivariate Point Outliers

Ver. 1.1.0 (2024-04-26)

This module provides a multivariate benchmark stream with configurable baselines per feature and additional random point outliers.

```
class mlpro.bf.streams.streams.point_outliers.StreamMLProOutliers(p_num_dim: int = 4,  
                                                                    p_num_instances: int =  
1000, p_functions: list[str] =  
['sin', 'cos', 'const', 'lin'],  
p_outlier_rate: float = 0.05,  
p_seed=None,  
p_logging=True,  
**p_kwargs)
```

Bases: StreamMLProBase

This benchmark stream provides multidimensional instances with configurable baselines per feature. Additionally, random point outliers per feature are induced.

### **p\_num\_dim**

[int] The number of dimensions or features of the data. Default = 3.

### **p\_num\_instances**

[int] Total number of instances. The value '0' means indefinite. Default = 1000.

### **p\_functions**

[list[str]] List of mathematical functions per feature.

### **p\_outlier\_rate**

[float] A value in [0,1] that defines the number of random outliers in % per feature.

### **p\_seed**

Seeding value for the random generator. Default = None (no seeding).

### **p\_logging**

Log level (see constants of class Log). Default: Log.C\_LOG\_ALL.

**C\_ID** = 'PointOutliersND'

**C\_NAME** = 'Point Outliers N-Dim'

**C\_TYPE** = 'Benchmark'

**C\_VERSION** = '1.1.0'

**C\_SCIREF\_ABSTRACT** = 'This benchmark stream provides multidimensional instances with configurable baselines per feature. Additionally, random point outliers per feature are induced.'

**C\_BOUNDARIES** = [0, 0]

**\_setup\_feature\_space()** → *MSpace*

Custom method to set up the feature space of the stream. It is called by method get\_feature\_space().

### **Returns**

**feature\_space** – Feature space of the stream.

### **Return type**

*MSpace*

**`_init_dataset()`**

Custom method to generate stream data as a numpy array named `self._dataset`.

**`_get_next()`** → *Instance*

Custom method to determine the next data stream instance. At the end of the stream exception `StopIteration` is to be raised. See method `__next__()` for more details.

**Returns**

**instance** – Next instance of data stream or `None`.

**Return type**

*Instance*

**`_fct_sin(p_x, p_outlier: bool)`**

**`_fct_cos(p_x, p_outlier: bool)`**

**`_fct_const(p_x, p_outlier: bool)`**

**`_fct_lin(p_x, p_outlier: bool)`**

**Random Point Clouds (2D, 3D, ND)**

Ver. 1.2.2 (2024-02-09)

This module provides the native stream classes `StreamMLProClouds`, `StreamMLProClouds2D4C1000Static`, `StreamMLProClouds3D8C2000Static`, `StreamMLProClouds2D4C5000Dynamic` and `StreamMLProClouds3D8C10000Dynamic`. These stream provides instances with `self.C_NUM_DIMENSIONS` dimensional random feature data, placed around centers (can be defined by user) which may or maynot move over time.

```
class mlpro.bf.streams.streams.clouds.StreamMLProClouds(p_num_dim: int = 3, p_num_instances: int
= 1000, p_num_clouds: int = 4, p_radii:
list = [100.0], p_weights: list = [],
p_velocity: float = 0.0, p_seed=None,
p_logging=True, **p_kwargs)
```

Bases: `StreamMLProBase`

This benchmark stream class generates freely configurable random point clouds of any number, size and dimensionality. Optionally, the centers of the clouds are static or in motion.

**Parameters**

- **`p_num_dim (int)`** – The number of dimensions or features of the data. Default = 3.
- **`p_num_instances (int)`** – Total number of instances. The value '0' means indefinite. Default = 1000.
- **`p_num_clouds (int)`** – Number of clouds. Default = 4.
- **`p_radii (list)`** – Radii of the clouds. Default = 100.
- **`p_weights (list [])`** – Optional list of integer weights per cloud. For example, a list [1,2] causes the second cloud to be flooded with two times more instances than the first one. If empty or `None`, all clouds are flooded randomly but equally.
- **`p_velocity (float)`** – Velocity for the centers in unit 1/di. Default = 0.0.
- **`p_seed`** – Seeding value for the random generator. Default = `None` (no seeding).
- **`p_logging`** – Log level (see constants of class `Log`). Default: `Log.C_LOG_ALL`.

```
C_ID = 'CloudsNDim'
C_NAME = 'Clouds N-Dim'
C_TYPE = 'Benchmark'
C_VERSION = '1.0.0'
C_SCIREF_ABSTRACT = 'Demo stream provides self.C_NUM_INSTANCES
C_NUM_DIMENSIONS-dimensional instances per cluster randomly positioned around
centers which may or maynot move over time.'
C_BOUNDARIES = [-1000, 1000]
```

**\_setup\_feature\_space()** → *MSpace*

Custom method to set up the feature space of the stream. It is called by method `get_feature_space()`.

**Returns**

**feature\_space** – Feature space of the stream.

**Return type**

*MSpace*

**\_init\_dataset()**

Custom method to generate stream data as a numpy array named `self._dataset`.

**\_get\_next()** → *Instance*

Custom method to determine the next data stream instance. At the end of the stream exception `StopIteration` is to be raised. See method `__next__()` for more details.

**Returns**

**instance** – Next instance of data stream or `None`.

**Return type**

*Instance*

```
class mlpro.bf.streams.streams.clouds.StreamMLProClouds2D4C1000Static(p_radii: list = [20.0],
                                                                    p_logging=True,
                                                                    **p_kwargs)
```

Bases: *StreamMLProClouds*

This benchmark stream generates 1000 2-dimensional instances that form 4 static random point clouds.

See also: class `StreamMLProClouds`

**Parameters**

- **p\_radii** (*list*) – Radii of the clouds. Default = 20.
- **p\_seed** – Seeding value for the random generator. Default = `None` (no seeding).
- **p\_logging** – Log level (see constants of class `Log`). Default: `Log.C_LOG_ALL`.

```
C_ID = 'Clouds2D4C1000Static'
```

```
C_NAME = 'Static Clouds 2D'
```

```
C_VERSION = '1.0.1'
```

```
C_NUM_DIMENSIONS = 2
```

```
C_NUM_INSTANCES = 1000
```



```
C_SCIREF_ABSTRACT = 'Demo stream provides 1000 2D instances randomly positioned
around four fixed centers.'
```

```
C_BOUNDARIES = [-100, 100]
```

```
class mlpro.bf.streams.streams.clouds.StreamMLProClouds3D8C2000Static(p_radii: list = [20.0],
                                                                    p_logging=True,
                                                                    **p_kwargs)
```

Bases: [StreamMLProClouds](#)

This benchmark stream generates 2000 3-dimensional instances that form 8 static random point clouds.

See also: class StreamMLProClouds

#### Parameters

- **p\_radii** (*list*) – Radii of the clouds. Default = 20.
- **p\_seed** – Seeding value for the random generator. Default = None (no seeding).
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL.

```
C_ID = 'Clouds3D8C2000Static'
```

```
C_NAME = 'Static Clouds 3D'
```

```
C_VERSION = '1.0.1'
```

```
C_NUM_DIMENSIONS = 3
```

```
C_NUM_INSTANCES = 2000
```

```
C_SCIREF_ABSTRACT = 'Demo stream provides 2000 3D instances randomly positioned
around eight fixed centers.'
```

```
C_BOUNDARIES = [-100, 100]
```

```
class mlpro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dynamic(p_radii: list = [100.0],
                                                                    p_velocity: float = 1,
                                                                    p_logging=True,
                                                                    **p_kwargs)
```

Bases: [StreamMLProClouds](#)

This benchmark stream generates 5000 2-dimensional instances that form 4 dynamic random point clouds.

See also: class StreamMLProClouds

#### Parameters

- **p\_radii** (*list*) – Radii of the clouds. Default = 100.
- **p\_seed** – Seeding value for the random generator. Default = None (no seeding).
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL.

```
C_ID = 'Clouds2D4C5000Dynamic'
```

```
C_NAME = 'Dynamic Clouds 2D'
```

```
C_VERSION = '1.0.1'
```

```
C_NUM_DIMENSIONS = 2
```

```
C_NUM_INSTANCES = 5000
```

```
C_SCIREF_ABSTRACT = 'Demo stream provides 2000 2D instances randomly positioned  
around four randomly moving centers.'
```

```
C_BOUNDARIES = [-1000, 1000]
```

```
class mlpro.bf.streams.streams.clouds.StreamMLProClouds3D8C100000Dynamic(p_radii: list =  
    [100.0], p_velocity:  
    float = 1,  
    p_logging=True,  
    **p_kwargs)
```

Bases: [StreamMLProClouds](#)

This benchmark stream generates 10000 3-dimensional instances that form 8 dynamic random point clouds.

See also: class [StreamMLProClouds](#)

#### Parameters

- **p\_radii** (*list*) – Radii of the clouds. Default = 100.
- **p\_seed** – Seeding value for the random generator. Default = None (no seeding).
- **p\_logging** – Log level (see constants of class [Log](#)). Default: [Log.C\\_LOG\\_ALL](#).

```
C_ID = 'Clouds3D8C100000Dynamic'
```

```
C_NAME = 'Dynamic Clouds 3D'
```

```
C_VERSION = '1.0.1'
```

```
C_NUM_DIMENSIONS = 3
```

```
C_NUM_INSTANCES = 10000
```

```
C_SCIREF_ABSTRACT = 'Demo stream provides 10000 3D instances randomly positioned  
around eight randomly moving centers.'
```

```
C_BOUNDARIES = [-1000, 1000]
```

## Stream Tasks

### Deriver

Ver. 1.0.0 (2023-02-05)

This module provides a stream task class [Deriver](#) to derive the data of instances.

```
class mlpro.bf.streams.tasks.deriver.Deriver(p_name: str = None, p_range_max=1, p_duplicate_data:  
    bool = False, p_visualize: bool = False, p_logging=True,  
    p_features: list = None, p_labels: list = None,  
    p_derived_feature: Feature = None, p_derived_label:  
    Label = None, p_order_derivative: int = 1, **p_kwargs)
```

Bases: [StreamTask](#)

This stream task extend the feature and label data of incoming instances with a derivation of a pre-selected feature.

**Parameters**

- **p\_name** (*str*) – Optional name of the task. Default is None.
- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class Range. Default is Range.C\_RANGE\_PROCESS.
- **p\_duplicate\_data** (*bool*) – If True, instances will be duplicated before processing. Default = False.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL
- **p\_features** (*list*) – The list of current features in the stream. Default = None.
- **p\_labels** (*list*) – The list of current labels in the stream. Default = None.
- **p\_derived\_feature** (*Feature*) – A pre-selected feature that would like to be derived. Default = None.
- **p\_derived\_label** (*Feature*) – A correspondig label of the derived feature. Default = None.
- **p\_order\_derivative** (*int*) – The derivative of order n. Default = None.
- **p\_kwargs** (*dict*) – Further optional named parameters.

**C\_NAME** = 'Deriver'

**C\_PLOT\_STANDALONE**: bool = True

**\_derive\_data**(*p\_inst*: *Instance*)

**\_run**(*p\_inst\_new*: set, *p\_inst\_del*: set)

Custom method that is called by method run().

**Parameters**

- **p\_inst\_new** (set) – Set of new stream instances to be processed.
- **p\_inst\_del** (set) – Set of obsolete stream instances to be removed.

```
class mlpro.bf.streams.tasks.deriver.DerivativeFunction(p_name: str, p_id: int = None, p_type: int
    = None, p_unit_in: str = None, p_unit_out:
    str = None, p_dt: float = 0.01,
    p_logging=True, **p_args)
```

Bases: *TransferFunction*

**\_set\_function\_parameters**(*p\_args*) → bool

This method provides a functionality to set the parameters of the transfer function.

**Parameters**

**p\_args** (*dict*) – set of parameters of the transfer function.

**Returns**

true means no parameters are missing.

**Return type**

bool

**\_custom\_function**(*p\_input*, *p\_range*)

This function represents the template to create a custom function and must be redefined.

**Parameters**

- **p\_input** – input value.
- **p\_range** – range of the calculation. None means 0. Default: None.

**Returns**

output value.

**Return type**

float

## Rearranger

Ver. 1.0.3 (2022-12-19)

This module provides a stream task class Rearranger to rearrange the feature and label space of instances.

```
class mlpro.bf.streams.tasks.rearranger.Rearranger(p_name: str = None, p_range_max=1,
                                                  p_duplicate_data: bool = False, p_visualize: bool
                                                  = False, p_logging=True, p_features_new: list =
                                                  [], p_labels_new: list = [], **p_kwargs)
```

Bases: [StreamTask](#)

This stream task rearrange the feature and/or label data of incoming instances. To this regard, two additional parameters `p_features_new` and `p_labels_new` describe the dimensions of the feature/label space of the resulting instances.

**Parameters**

- **p\_name** (*str*) – Optional name of the task. Default is None.
- **p\_range\_max** (*int*) – Maximum range of asynchronicity. See class Range. Default is `Range.C_RANGE_PROCESS`.
- **p\_duplicate\_data** (*bool*) – If True, instances will be duplicated before processing. Default = False.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default: `Log.C_LOG_ALL`
- **p\_features\_new** (*list*) – List of resulting features that are described as tuples ( 'F' or 'L', list[Dimension] ). The first component specifies the origin ( 'F' = feature space, 'L' = label space). The second component is a list of dimension objects.
- **p\_labels\_new** (*list[Label]*) – List of resulting labels that are described as tuples ( 'F' or 'L', list[Dimension] ).
- **p\_kwargs** (*dict*) – Further optional named parameters.

**C\_NAME** = 'Rearranger'

**C\_PLOT\_STANDALONE**: bool = True

**\_prepare\_rearrangement**(p\_inst: [Instance](#))

**\_rearrange**(p\_inst: [Instance](#))

**\_run**(p\_inst\_new: set, p\_inst\_del: set)

Custom method that is called by method run().

**Parameters**

- **p\_inst\_new** (set) – Set of new stream instances to be processed.
- **p\_inst\_del** (set) – Set of obsolete stream instances to be removed.

## Windows

Ver. 1.1.5 (2023-02-02)

This module provides pool of window objects further used in the context of online adaptivity.

```
class mlpro.bf.streams.tasks.windows.Window(p_buffer_size: int, p_delay: bool = False,
                                             p_enable_statistics: bool = False, p_name: str = None,
                                             p_range_max=1, p_duplicate_data: bool = False,
                                             p_visualize: bool = False, p_logging=True, **p_kwargs)
```

Bases: [StreamTask](#)

This is the base class for window implementations

### Parameters

- **p\_buffer\_size** (int) – the size/length of the buffer/window.
- **p\_delay** (bool, optional) – Set to true if full buffer is desired before passing the window data to next step. Default is false.
- **p\_name** (str, optional) – Name of the Window. Default is None.
- **-Optional** (p\_logging) – Maximum range of task parallelism for window task. Default is set to multithread.
- **p\_duplicate\_data** (bool) – If True, instances will be duplicated before processing. Default = False.
- **p\_ada** (bool, optional) – Adaptivity property of object. Default is True.
- **-Optional** – Log level for the object. Default is log everything.

**C\_NAME** = 'Window'

**C\_PLOT\_STANDALONE**: bool = False

**C\_PLOT\_IN\_WINDOW** = 'In Window'

**C\_PLOT\_OUTSIDE\_WINDOW** = 'Out Window'

**C\_EVENT\_BUFFER\_FULL** = 'BUFFER\_FULL'

**C\_EVENT\_DATA\_REMOVED** = 'DATA\_REMOVED'

**\_run**(p\_inst\_new: list, p\_inst\_del: list)

Method to run the window including adding and deleting of elements

### Parameters

- **p\_inst\_new** (list) – Instance/s to be added to the window
- **p\_inst\_del** (list) – Instance/s to be deleted from the window

**get\_buffered\_data**()

Method to fetch the data from the window buffer

### Returns

- **buffer** (*dict*) – the buffered data in the form of dictionary
- **buffer\_pos** (*int*) – the latest buffer position

**get\_boundaries()**

Method to get the current boundaries of the Window

**Returns**

**boundaries** – Returns the current window boundaries in the form of a Numpy array.

**Return type**

np.ndarray

**get\_mean()**

Method to get the mean of the data in the Window.

**Returns**

**mean** – Returns the mean of the current data in the window in the form of a Numpy array.

**Return type**

np.ndarray

**get\_variance()**

Method to get the variance of the data in the Window.

**Returns**

**variance** – Returns the variance of the current data in the window as a numpy array.

**Return type**

np.ndarray

**get\_std\_deviation()**

Method to get the standard deviation of the data in the window.

**Returns**

**std** – Returns the standard deviation of the data in the window as a numpy array.

**Return type**

np.ndarray

**\_init\_plot\_2d**(*p\_figure: Figure*, *p\_settings: PlotSettings*)

Custom method to initialize a 2D plot for the window object

**Parameters**

- **p\_figure** (*Figure*) – The figure object that hosts the plot
- **p\_settings** (*list of PlotSettings objects.*) – Additional settings for the plot

**\_init\_plot\_3d**(*p\_figure: Figure*, *p\_settings: PlotSettings*)

Custom method to initialize a 3D plot for window object

**Parameters**

- **p\_figure** (*matplotlib.figure.Figure*) – The figure object to host the plot.
- **p\_settings** (*PlotSettings*) – Additional Settings for the plot

**\_init\_plot\_nd**(*p\_figure: Figure*, *p\_settings: PlotSettings*)

Custom method to initialize plot for Window tasks for N-dimensional plotting.

**Parameters**

- **p\_figure** (*Figure*) – Figure to host the plot

- **p\_settings** ([PlotSettings](#)) – PlotSettings objects with specific settings for the plot

**\_update\_plot\_2d**(*p\_settings*: [PlotSettings](#), *p\_inst\_new*: list, *p\_inst\_del*: list, *\*\*p\_kwargs*)

Default 3-dimensional plotting implementation for window tasks. See class `mlpro.bf.plot.Plottable` for more details.

#### Parameters

- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.
- **p\_inst\_new** (list) – List of new stream instances to be plotted.
- **p\_inst\_del** (list) – List of obsolete stream instances to be removed.
- **p\_kwargs** (dict) – Further optional plot parameters.

**\_update\_plot\_3d**(*p\_settings*: [PlotSettings](#), *p\_inst\_new*: list, *p\_inst\_del*: list, *\*\*p\_kwargs*)

Default 3-dimensional plotting implementation for window tasks. See class `mlpro.bf.plot.Plottable` for more details.

#### Parameters

- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.
- **p\_inst\_new** (list) – List of new stream instances to be plotted.
- **p\_inst\_del** (list) – List of obsolete stream instances to be removed.
- **p\_kwargs** (dict) – Further optional plot parameters.

**\_update\_plot\_nd**(*p\_settings*: [PlotSettings](#), *p\_inst\_new*: list, *p\_inst\_del*: list, *\*\*p\_kwargs*)

Default N-dimensional plotting implementation for window tasks. See class `mlpro.bf.plot.Plottable` for more details.

#### Parameters

- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.
- **p\_inst\_new** (list) – List of new stream instances to be plotted.
- **p\_inst\_del** (list) – List of obsolete stream instances to be removed.
- **p\_kwargs** (dict) – Further optional plot parameters.

## Samplers

### Min-Wise Samplers

Ver. 1.0.0 (2023-04-16)

This module provides a ready-to-use stream sampler class `SamplerMinWise`, in which a set of instances is classified as one cluster and each of them is weighted with a random uniform value from 0 to 1. The smallest weighted instance in the cluster is not omitted, while the other is omitted. This algorithm is proposed by Suman Nath, et al.

**class** `mlpro.bf.streams.samplers.min_wise.SamplerMinWise`(*p\_num\_instances*: int = 0, *p\_cluster\_size*: float = 1, *p\_seed*: int = 0)

Bases: [Sampler](#)

A ready-to-use class for data streams with min-wise sampler. This object can be used in `Stream`.

#### Parameters

- **p\_num\_instances** (*int*) – Number of instances. This parameter has no affect in this sampler method. Default = 0.
- **p\_cluster\_size** (*int*) – Number of instances in a cluster. Default = 10.
- **p\_seed** (*int*) – Random seeding. Default = 0.

**C\_TYPE** = 'Min-Wise Sampler'

**C\_SCIREF\_TYPE\_PROCEEDINGS** = 'Proceedings'

**C\_SCIREF\_TYPE** = 'Proceedings'

**C\_SCIREF\_AUTHOR** = 'Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson'

**C\_SCIREF\_TITLE** = 'Synopsis Diffusion for Robust Aggregation in Sensor Networks'

**C\_SCIREF\_YEAR** = '2004'

**C\_SCIREF\_ISBN** = '1581138792'

**C\_SCIREF\_PUBLISHER** = 'Association for Computing Machinery'

**C\_SCIREF\_URL** = 'https://doi.org/10.1145/1031495.1031525'

**C\_SCIREF\_DOI** = '10.1145/1031495.1031525'

**C\_SCIREF\_BOOKTITLE** = 'Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems'

**C\_SCIREF\_PAGES** = '250-262'

**reset()**

A method to reset the sampler's settings.

**\_omit\_instance**(*p\_inst*: [Instance](#)) → bool

A custom method to filter any incoming instances, which is being called by omit\_instance() method.

**Parameters**

**p\_inst** ([Instance](#)) – An input instance to be filtered.

**Returns**

False means the input instance is not omitted, otherwise True.

**Return type**

bool

## Random Samplers

Ver. 1.0.0 (2023-04-14)

This module provides a ready-to-use stream sampler class `SamplerRND`.

```
class mlpro.bf.streams.samplers.random.SamplerRND(p_num_instances: int = 0, p_max_step_rate: int = 5, p_seed: int = 0)
```

Bases: [Sampler](#)

A ready-to-use class for data streams with random sampler. This object can be used in Stream.

**Parameters**



- **p\_num\_instances** (*int*) – Number of instances. This parameter has no affect in this sampler method. Default = 0.
- **p\_max\_step\_rate** (*int*) – Maximum step rate parameter for non time series data streams. Default = 5.
- **p\_seed** (*int*) – Random seeding. Default = 0.

**C\_TYPE** = 'Random Sampler'

**reset()**

A method to reset the sampler's settings.

**\_omit\_instance**(*p\_inst*: [Instance](#)) → bool

A custom method to filter any incoming instances, which is being called by omit\_instance() method.

#### Parameters

**p\_inst** ([Instance](#)) – An input instance to be filtered.

#### Returns

False means the input instance is not omitted, otherwise True.

#### Return type

bool

## Reservoir Samplers with Algorithm R

Ver. 1.0.0 (2023-04-16)

This module provides a ready-to-use stream sampler class `SamplerReservoir`. Reservoir sampling is a simple algorithm that is still part of a random sampling algorithm. This algorithm is proposed by Jeffrey Vitter. In this module, we apply the default algorithm of reservoir sampling with Algorithm R. However, we enhance the algorithm, where the `p_num_instances` remains unknown.

```
class mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir(p_num_instances: int =
                                                                    None, p_reservoir_size: int
                                                                    = 10, p_seed: int = 0)
```

Bases: [Sampler](#)

A ready-to-use class for data streams with reservoir sampler using algorithm R. This object can be used in Stream.

#### Parameters

- **p\_num\_instances** (*int*) – Number of instances. This parameter is optional. Default = None.
- **p\_reservoir\_size** (*int*) – Size of an reservoir. Default = 10.
- **p\_seed** (*int*) – Random seeding. Default = 0.

**C\_TYPE** = 'Reservoir Sampler (Algorithm R)'

**C\_SCIREF\_TYPE\_ARTICLE** = 'Journal Article'

**C\_SCIREF\_TYPE** = 'Journal Article'

**C\_SCIREF\_AUTHOR** = 'Jeffrey S. Vitter'

**C\_SCIREF\_TITLE** = 'Random Sampling with a Reservoir'

```
C_SCIREF_YEAR = '1985'
C_SCIREF_PUBLISHER = 'Association for Computing Machinery'
C_SCIREF_VOLUME = '11'
C_SCIREF_NUMBER = '1'
C_SCIREF_URL = 'https://doi.org/10.1145/3147.3165'
C_SCIREF_DOI = '10.1145/3147.3165'
C_SCIREF_JOURNAL = 'ACM Trans. Math. Softw.'
C_SCIREF_MONTH = 'Mar'
C_SCIREF_PAGES = '37-57'
```

**reset()**

A method to reset the sampler's settings.

**\_omit\_instance**(*p\_inst*: [Instance](#)) → bool

A custom method to filter any incoming instances, which is being called by omit\_instance() method.

**Parameters**

**p\_inst** ([Instance](#)) – An input instance to be filtered.

**Returns**

False means the input instance is not omitted, otherwise True.

**Return type**

bool

## Weighted Random Samplers

Ver. 1.0.0 (2023-04-16)

This module provides a ready-to-use stream sampler class `SamplerWeightedRND`, in which each instance is randomly uniformly weighted. Then, it is compared to a pre-defined threshold. If the weight of an instance is higher than the threshold, then the instance is not omitted. Otherwise, it is omitted.

```
class mlpro.bf.streams.samplers.weighted_random.SamplerWeightedRND(p_num_instances: int = 0,
                                                                    p_threshold: float = 0.5,
                                                                    p_seed: int = 0)
```

Bases: [Sampler](#)

A ready-to-use class for data streams with random sampler and weighted instance. This object can be used in Stream.

**Parameters**

- **p\_num\_instances** (*int*) – Number of instances. This parameter has no affect in this sampler method. Default = 0.
- **p\_threshold** (*float*) – Threshold for selection of an instance. This value must be between 0 to 1. Default = 0.5.
- **p\_seed** (*int*) – Random seeding. Default = 0.

```
C_TYPE = 'Weighted Random Sampler'
```

**reset()**

A method to reset the sampler's settings.

**\_omit\_instance**(*p\_inst*: [Instance](#)) → bool

A custom method to filter any incoming instances, which is being called by omit\_instance() method.

**Parameters**

**p\_inst** ([Instance](#)) – An input instance to be filtered.

**Returns**

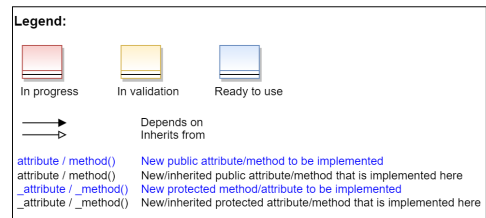
False means the input instance is not omitted, otherwise True.

**Return type**

bool

**BF-SYSTEMS - Systems**

## 452



The Double Pendulum System is an implementation of a classic control problem of Double Pendulum system. The dynamics of the system are based on the [Double Pendulum](#) implementation by [Matplotlib](#). The double pendulum is a system of two poles, with the inner pole connected to a fixed point at one end and to outer pole at other end. The native implementation of Double Pendulum consists of an input motor providing the torque in either directions to actuate the system.

```
class mlpro.bf.systems.pool.doublependulum.DoublePendulumSystemRoot(p_id=None, p_name: str =
                                                                    None, p_range_max: int =
                                                                    0, p_autorun=0,
                                                                    p_class_shared=None,
                                                                    p_mode=0,
                                                                    p_latency=None,
                                                                    p_t_step=None,
                                                                    p_max_torque=20,
                                                                    p_l1=1.0, p_l2=1.0,
                                                                    p_m1=1.0, p_m2=1.0,
                                                                    p_init_angles='random',
                                                                    p_g=9.8, p_fct_strans:
                                                                    FctSTrans = None,
                                                                    p_fct_success: FctSuccess
                                                                    = None, p_fct_broken:
                                                                    FctBroken = None,
                                                                    p_mujoco_file=None,
                                                                    p_frame_skip=None,
                                                                    p_state_mapping=None,
                                                                    p_action_mapping=None,
                                                                    p_camera_conf=None,
                                                                    p_history_length=5,
                                                                    p_visualize: bool = False,
                                                                    p_random_range: list =
                                                                    None, p_balancing_range:
                                                                    list = (-0.2, 0.2),
                                                                    p_swinging_outer_pole_range=(0.2,
                                                                    0.5), p_break_swinging:
                                                                    bool = False,
                                                                    p_logging=True,
                                                                    **p_kwargs)
```

Bases: [System](#)

This is the root double pendulum environment class inherited from Environment class with four dimensional state space and underlying implementation of the Double Pendulum dynamics, default reward strategy.

#### Parameters

- **p\_mode** – Mode of environment. Possible values are Mode.C\_MODE\_SIM(default) or Mode.C\_MODE\_REAL.
- **p\_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C\_LATENCY is used by default.
- **p\_max\_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p\_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p\_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p\_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5

- **p\_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25
- **p\_init\_angles** (*str, optional*) – C\_ANGLES\_UP starts the pendulum in an upright position C\_ANGLES\_DOWN starts the pendulum in a downward position C\_ANGLES\_RND starts the pendulum from a random position.
- **p\_g** (*float, optional*) – Gravitational acceleration. The default is 9.8
- **p\_history\_length** (*int, optional*) – Historical trajectory points to display. The default is 5.
- **p\_fct\_strans** (*FctSTrans, optional*) – The custom State transition function.
- **p\_fct\_success** (*FctSuccess, optional*) – The custom Success Function.
- **p\_fct\_broken** (*FctBroken, optional*) – The custom Broken Function.
- **p\_mujoco\_file** (*optional*) – The corresponding mujoco file
- **p\_frame\_skip** (*optional*) – Number of frames to be skipped for visualization.
- **p\_state\_mapping** (*optional*) – State mapping configurations.
- **p\_action\_mapping** (*optional*) – Action mapping configurations.
- **p\_camera\_conf** (*optional*) – Camera configurations for mujoco specific visualization.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_random\_range** (*list*) – The boundaries for state space for initialization of environment randomly
- **range** (*p\_balancing*) – The boundaries for state space of environment in balancing region
- **p\_swinging\_outer\_pole\_range** – The boundaries for state space of environment in swinging of outer pole region
- **p\_break\_swinging** (*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p\_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C\_LOG\_WE.

```
C_NAME = 'DoublePendulumSystemRoot'
```

```
C_SCIREF_TYPE = 'Online'
```

```
C_SCIREF_AUTHOR = 'John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the  
Matplotlib development team'
```

```
C_SCIREF_TITLE = 'The Double Pendulum Problem'
```

```
C_SCIREF_URL =  
'https://matplotlib.org/stable/gallery/animation/double_pendulum.html'
```

```
C_PLOT_ACTIVE: bool = True
```

```
C_PLOT_DEFAULT_VIEW: str = '2D'
```

```
C_CYCLE_LIMIT = 0
```

```
C_LATENCY = datetime.timedelta(microseconds=40000)
```

```
C_ANGLES_UP = 'up'
```

`C_ANGLES_DOWN = 'down'`

`C_ANGLES_RND = 'random'`

`C_VALID_ANGLES = ['up', 'down', 'random']`

`C_THRSH_GOAL = 0`

`C_ANI_FRAME = 30`

`C_ANI_STEP = 0.001`

`setup_spaces()`

Method to setup the spaces for the Double Pendulum root environment. This method sets up four dimensional Euclidean space for the root DP environment.

`_reset(p_seed=None) → None`

This method is used to reset the environment. The environment is reset to the initial position set during the initialization of the environment.

**Parameters**

**p\_seed** (*int*, *optional*) – The default is None.

`_derivs(p_state, t, p_torque)`

This method is used to calculate the derivatives of the system, given the current states.

**Parameters**

- **state** (*list*) – list of current state elements [theta 1, omega 1, acc 1, theta 2, omega 2, acc 2]
- **t** (*list*) – current Timestep
- **torque** (*float*) – Applied torque of the motor

**Returns**

**dydx** – The derivatives of the given state

**Return type**

list

`_simulate_reaction(p_state: State, p_action: Action)`

This method is used to calculate the next states of the system after a set of actions.

**Parameters**

**p\_state** (*State*) – current State. **p\_action** : Action  
current Action.

**Returns**

**\_state** – Current states after the simulation of latest action on the environment.

**Return type**

*State*

`_compute_broken(p_state: State) → bool`

Custom method to compute broken state. In this case always returns false as the environment doesn't break

`_compute_success(p_state: State)`

Custom method to return the success state of the environment based on the distance between current state, goal state and the goal threshold parameter

**Parameters**

**p\_state** ([State](#)) – current state of the environment

**Returns**

True if the distance between current state and goal state is less than the goal threshold else  
false

**Return type**

bool

**`_normalize(p_state: list)`**

Custom method to normalize the State values of the DP env based on static boundaries provided by MLPro

**Parameters**

**p\_state** ([State](#)) – The state to be normalized

**Returns**

Normalized state values

**Return type**

state

**`_init_plot_2d(p_figure: Figure, p_settings: PlotSettings)`**

Custom method to initialize a 2D plot. If attribute `p_settings.axes` is not None the initialization shall be done there. Otherwise a new Matplotlib Axes object shall be created in the given figure and stored in `p_settings.axes`.

**Parameters**

- **p\_figure** (*Matplotlib.figure.Figure*) – Matplotlib figure object to host the sub-plot(s).
- **p\_settings** ([PlotSettings](#)) – Object with further plot settings.

**`_update_plot_2d(p_settings: PlotSettings, **p_kwargs)`**

This method updates the plot figure of each episode. When the figure is detected to be an embedded figure, this method will only set up the necessary data of the figure.



```

class mlpro.bf.systems.pool.doublependulum.DoublePendulumSystemS4(p_id=None, p_name: str =
                                                                    None, p_range_max: int = 0,
                                                                    p_autorun=0,
                                                                    p_class_shared=None,
                                                                    p_mode=0, p_latency=None,
                                                                    p_t_step=None,
                                                                    p_max_torque=20, p_l1=1.0,
                                                                    p_l2=1.0, p_m1=1.0,
                                                                    p_m2=1.0,
                                                                    p_init_angles='random',
                                                                    p_g=9.8, p_fct_strans:
                                                                    FctSTrans = None,
                                                                    p_fct_success: FctSuccess =
                                                                    None, p_fct_broken:
                                                                    FctBroken = None,
                                                                    p_mujoco_file=None,
                                                                    p_frame_skip=None,
                                                                    p_state_mapping=None,
                                                                    p_action_mapping=None,
                                                                    p_camera_conf=None,
                                                                    p_history_length=5,
                                                                    p_visualize: bool = False,
                                                                    p_random_range: list = None,
                                                                    p_balancing_range: list =
                                                                    (-0.2, 0.2),
                                                                    p_swinging_outer_pole_range=(0.2,
                                                                    0.5), p_break_swinging: bool
                                                                    = False, p_logging=True,
                                                                    **p_kwargs)

```

Bases: [DoublePendulumSystemRoot](#)

This is the Double Pendulum Static 4 dimensional environment that inherits from the double pendulum root class, inheriting the dynamics and default reward strategy.

#### Parameters

- **p\_mode** – Mode of environment. Possible values are Mode.C\_MODE\_SIM(default) or Mode.C\_MODE\_REAL.
- **p\_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C\_LATENCY is used by default.
- **p\_max\_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p\_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p\_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p\_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p\_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25
- **p\_init\_angles** (*str, optional*) – C\_ANGLES\_UP starts the pendulum in an upright position C\_ANGLES\_DOWN starts the pendulum in a downward position C\_ANGLES\_RND starts the pendulum from a random position.
- **p\_g** (*float, optional*) – Gravitational acceleration. The default is 9.8

- **p\_history\_length**(*int*, *optional*) – Historical trajectory points to display. The default is 5.
- **p\_visualize**(*bool*) – Boolean switch for visualisation. Default = False.
- **p\_plot\_level**(*int*) – Types and number of plots to be plotted. Default = ALL  
C\_PLOT\_DEPTH\_ENV only plots the environment C\_PLOT\_DEPTH\_REWARD only plots the reward C\_PLOT\_ALL plots both reward and the environment
- **p\_rst\_balancingL** – Reward strategy to be used for the balancing region of the environment
- **p\_rst\_swinging** – Reward strategy to be used for the swinging region of the environment
- **p\_reward\_weights**(*list*) – List of weights to be added to the dimensions of the state space for reward computation
- **p\_reward\_trend**(*bool*) – Boolean value stating whether to plot reward trend
- **p\_reward\_window**(*int*) – The number of latest rewards to be shown in the plot. Default is 0
- **p\_random\_range**(*list*) – The boundaries for state space for initialization of environment randomly
- **range**(*p\_balancing*) – The boundaries for state space of environment in balancing region
- **p\_break\_swinging**(*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p\_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C\_LOG\_WE.

**C\_NAME** = 'DoublePendulumSystemS4'

**\_normalize**(*p\_state: list*)

Method for normalizing the State values of the DP env based on MinMax normalisation based on static boundaries provided by MLPro.

**Parameters**

**p\_state** – The state to be normalized

**Returns**

Normalized state values

**Return type**

state

**\_obs\_to\_mujoco**(*p\_state*)

```

class mlpro.bf.systems.pool.doublependulum.DoublePendulumSystemS7(p_id=None, p_name: str =
                                                                    None, p_range_max: int = 0,
                                                                    p_autorun=0,
                                                                    p_class_shared=None,
                                                                    p_mode=0, p_latency=None,
                                                                    p_t_step=None,
                                                                    p_max_torque=20, p_l1=1.0,
                                                                    p_l2=1.0, p_m1=1.0,
                                                                    p_m2=1.0,
                                                                    p_init_angles='random',
                                                                    p_g=9.8, p_fct_strans:
                                                                    FctSTrans = None,
                                                                    p_fct_success: FctSuccess =
                                                                    None, p_fct_broken:
                                                                    FctBroken = None,
                                                                    p_mujoco_file=None,
                                                                    p_frame_skip=None,
                                                                    p_state_mapping=None,
                                                                    p_action_mapping=None,
                                                                    p_camera_conf=None,
                                                                    p_history_length=5,
                                                                    p_visualize: bool = False,
                                                                    p_random_range: list = None,
                                                                    p_balancing_range: list =
                                                                    (-0.2, 0.2),
                                                                    p_swinging_outer_pole_range=(0.2,
                                                                    0.5), p_break_swinging: bool
                                                                    = False, p_logging=True,
                                                                    **p_kwargs)

```

Bases: [DoublePendulumSystemS4](#)

This is the classic implementation of Double Pendulum with 7 dimensional state space including derived accelerations of both the poles and the input torque. The dynamics of the system are inherited from the Double Pendulum Root class.

#### Parameters

- **p\_mode** – Mode of environment. Possible values are Mode.C\_MODE\_SIM(default) or Mode.C\_MODE\_REAL.
- **p\_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C\_LATENCY is used by default.
- **p\_max\_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p\_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p\_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p\_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p\_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25
- **p\_init\_angles** (*str, optional*) – C\_ANGLES\_UP starts the pendulum in an upright position C\_ANGLES\_DOWN starts the pendulum in a downward position C\_ANGLES\_RND starts the pendulum from a random position.
- **p\_g** (*float, optional*) – Gravitational acceleration. The default is 9.8

- **p\_history\_length** (*int*, *optional*) – Historical trajectory points to display. The default is 5.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_plot\_level** (*int*) – Types and number of plots to be plotted. Default = ALL  
C\_PLOT\_DEPTH\_ENV only plots the environment C\_PLOT\_DEPTH\_REWARD only plots the reward C\_PLOT\_ALL plots both reward and the environment
- **p\_rst\_balancingL** – Reward strategy to be used for the balancing region of the environment
- **p\_rst\_swinging** – Reward strategy to be used for the swinging region of the environment
- **p\_reward\_weights** (*list*) – List of weights to be added to the dimensions of the state space for reward computation
- **p\_reward\_trend** (*bool*) – Boolean value stating whether to plot reward trend
- **p\_reward\_window** (*int*) – The number of latest rewards to be shown in the plot. Default is 0
- **p\_random\_range** (*list*) – The boundaries for state space for initialization of environment randomly
- **range** (*p\_balancing*) – The boundaries for state space of environment in balancing region
- **p\_break\_swinging** (*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p\_logging** – Log level (see constants of class `mlpro.bf.various.Log`). Default = `Log.C_LOG_WE`.

**C\_NAME** = 'DoublePendulumSystemS7'

#### **setup\_spaces()**

Method to set up the state and action spaces of the classic Double Pendulum Environment. Inheriting from the root class, this method adds 3 dimensions for accelerations and torque respectively.

**\_reset**(*p\_seed=None*) → None

This method is used to reset the environment.

##### **Parameters**

- **p\_seed** (*int*, *optional*) – The default is None.

**\_simulate\_reaction**(*p\_state: State*, *p\_action: Action*)

This method is used to calculate the next states of the system after a set of actions.

##### **Parameters**

- **p\_state** (*State*) – current State.
- **p\_action** (*Action*) – current Action.

##### **Returns**

Current states after simulating the latest action.

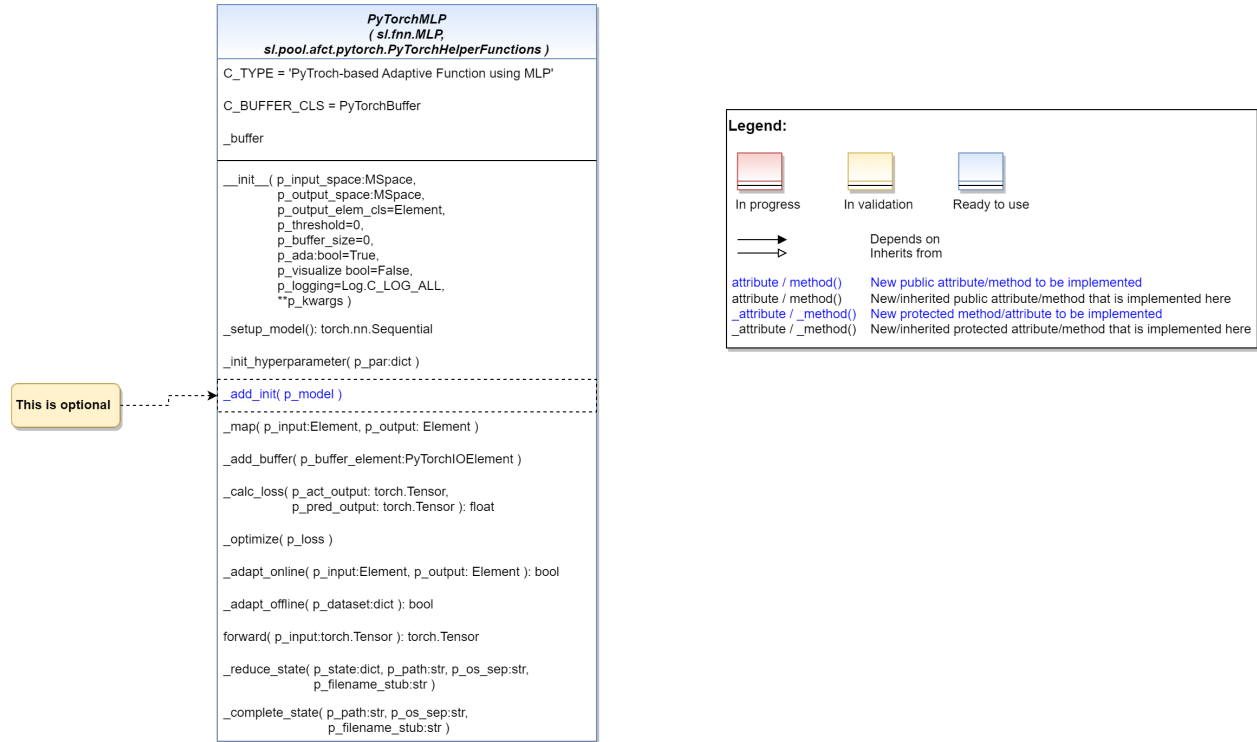
##### **Return type**

current\_state

## 11.2.2 MLPro-SL - Supervised Learning

### Adaptive Functions

#### PyTorch-based MLP



Ver. 1.2.4 (2023-07-14)

This module provides a template ready-to-use MLP model using PyTorch.

```
class mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP(p_input_space: ~mlpro.bf.math.basics.MSpace,
p_output_space: ~mlpro.bf.math.basics.MSpace,
p_output_elem_cls=<class
'mlpro.bf.math.basics.Element'>,
p_threshold=0, p_buffer_size=0, p_ada: bool =
True, p_visualize: bool = False,
p_logging=True, **p_kwargs)
```

Bases: [MLP](#), [PyTorchHelperFunctions](#)

Template class for an adaptive bi-multivariate mathematical function that adapts by supervised learning using PyTorch-based MLP.

#### Parameters

- **p\_input\_space** ([MSpace](#)) – Input space of function
- **p\_output\_space** ([MSpace](#)) – Output space of function
- **p\_output\_elem\_cls** – Output element class (compatible to/inherited from class [Element](#))
- **p\_threshold** (*float*) – Threshold for the difference between a setpoint and a computed output. Computed outputs with a difference less than this threshold will be assessed as 'good' outputs. Default = 0.

- **p\_buffer\_size** (*int*) – Initial size of internal data buffer. Default = 0 (no buffering).
- **p\_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** – Log level (see constants of class Log). Default: Log.C\_LOG\_ALL
- **p\_kwargs** (*Dict*) – Further model specific parameters (to be specified in child class).

**C\_TYPE = 'PyTorch-based Adaptive Function using MLP'**

**C\_BUFFER\_CLS**

alias of *PyTorchBuffer*

**\_setup\_model()** → Sequential

A method to set up a supervised learning network.

**Return type**

A set up supervised learning model

**\_init\_hyperparam(\*\*p\_par)**

A method to deal with the hyperparameters related to the MLP model.

### Hyperparameters

**p\_update\_rate :**

update rate.

**p\_num\_hidden\_layers :**

number of hidden layers.

**p\_output\_activation\_fct :**

extra activation function for the output layer.

**p\_optimizer :**

optimizer.

**p\_loss\_fct :**

loss function.

**p\_test\_data**

[float] the proportion of test data during the sampling process. Default = 0.3.

**p\_batch\_size**

[int] batch size of the buffer. Default = 100.

**p\_seed\_buffer**

[int] seeding of the buffer. Default = 1.

**p\_learning\_rate**

[float] learning rate of the optimizer. Default = 3e-4.

**p\_hidden\_size**

[int or list] number of hidden neurons. There are two possibilities to set up the hidden size: 1) if hidden\_size is an integer, then the number of neurons in all hidden layers are exactly the same. 2) if hidden\_size is in a list, then the user can define the number of neurons in each hidden layer, but make sure that the length of the list must be equal to the number of hidden layers.

**p\_activation\_fct**

[torch.nn or list] activation function. There are two possibilities to set up the activation function: 1)

if `activation_fct` is a single activation function, then the activation function after all hidden layers are exactly the same. 2) if `activation_fct` is in a list, then the user can define the activation function after each hidden layer, but make sure that the length of the list must be equal to the number of hidden layers.

**p\_weight\_bias\_init**

[bool, optional] weight and bias initialization. Default : True

**p\_weight\_init**

[torch.nn, optional] weight initialization function. Default : `torch.nn.init.orthogonal_`

**p\_bias\_init**

[torch.nn, optional] bias initialization function. Default : `lambda x: torch.nn.init.constant_(x, 0)`

**p\_gain\_init**

[int, optional] gain parameter of the weight and bias initialization. Default : `np.sqrt(2)`

**\_add\_init(p\_model)**

This method is optional and is intended for additional initialization process of the `_setup_model`.

**Parameters**

**p\_model** – model network

**Return type**

updated model network

**\_map(p\_input: Element, p\_output: Element)**

Maps a multivariate abscissa/input element to a multivariate ordinate/output element.

**Parameters**

- **p\_input** (`Element`) – Abscissa/input element object (type `Element`)
- **p\_output** (`Element`) – Setpoint ordinate/output element (type `Element`)

**\_add\_buffer(p\_buffer\_element: PyTorchIOElement)**

This method has a functionality to add data into the buffer.

**Parameters**

**p\_buffer\_element** (`PyTorchIOElement`) – An element of `PyTorchBuffer`.

**\_calc\_loss(p\_act\_output: Tensor, p\_pred\_output: Tensor) → float**

This method has a functionality to evaluate the adapted SL model.

**Parameters**

- **p\_act\_output** (`torch.Tensor`) – Actual output from the buffer.
- **p\_pred\_output** (`torch.Tensor`) – Predicted output by the SL model.

**\_optimize(p\_loss)**

This method provides provide a functionality to call the optimizer of the feedforward network.

**\_adapt\_online(p\_input: Element, p\_output: Element) → bool**

Adaptation mechanism for PyTorch based model for online learning.

**Parameters**

- **p\_input** (`Element`) – Abscissa/input element object (type `Element`)
- **p\_output** (`Element`) – Setpoint ordinate/output element (type `Element`)

**Return type**

bool

**`_adapt_offline`**(*p\_dataset: dict*) → bool

Adaptation mechanism for PyTorch based model for offline learning.

**Parameters**

**`p_dataset`** (*dict*) – a dictionary that consists of a set of data, which are splitted to 2 keys such as input and output. The value of each key is a torch.Tensor of the sampled data.

**Return type**

bool

**`forward`**(*p\_input: Tensor*) → Tensor

Forward propagation in neural networks to generate some output using PyTorch.

**Parameters**

**`p_input`** (*Element*) – Input data

**Returns**

**`output`** – Output data

**Return type**

*Element*

**`_reduce_state`**(*p\_state: dict, p\_path: str, p\_os\_sep: str, p\_filename\_stub: str*)

Custom method to reduce the given object state by components that can not be pickled. Further data files can be created in the given path and should use the given filename stub.

**Parameters**

- **`p_state`** (*dict*) – Object state dictionary to be reduced by components that can not be pickled.
- **`p_path`** (*str*) – Path to store further optional custom data files
- **`p_os_sep`** (*str*) – OS-specific path separator.
- **`p_filename_stub`** (*str*) – Filename stub to be used for further optional custom data files

**`_complete_state`**(*p\_path: str, p\_os\_sep: str, p\_filename\_stub: str*)

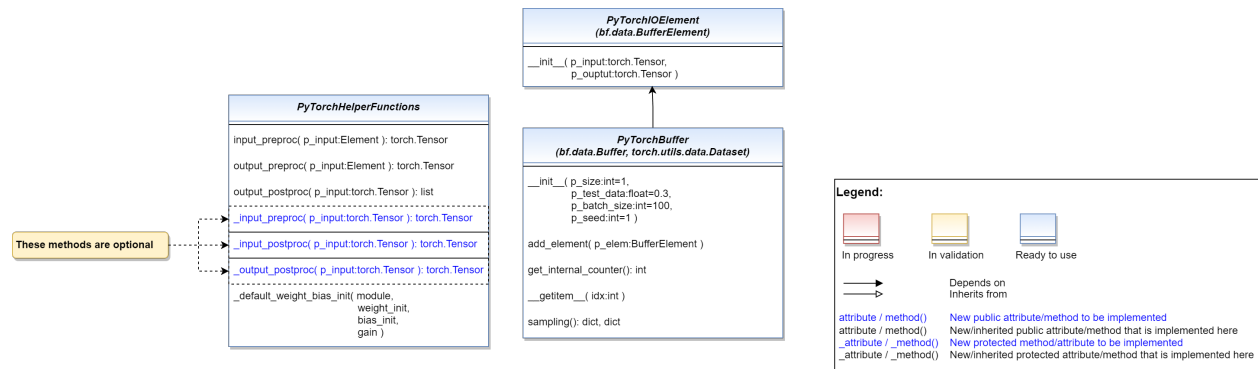
Custom method to complete the object state (=self) from external data sources. This method is called by standard method `__setstate__()` during unpickling the object from an external file.

**Parameters**

- **`p_path`** (*str*) – Path of the object pickle file (and further optional related files)
- **`p_os_sep`** (*str*) – OS-specific path separator.
- **`p_filename_stub`** (*str*) – Filename stub to be used for further optional custom data files



## PyTorch Helper Functions



Ver. 3.0.7 (2023-07-14)

This is a helper module for supervised learning models using PyTorch.

**class** mlpro.sl.pool.afct.pytorch.**PyTorchIOElement**(*p\_input: Tensor, p\_output: Tensor*)

Bases: BufferElement

This class provides a buffer element for PyTorch based SLNetwork.

### Parameters

- **p\_input** (Element) – Abscissa/input element object (type Element)
- **p\_output** (Element) – Setpoint ordinate/output element (type Element)

**class** mlpro.sl.pool.afct.pytorch.**PyTorchBuffer**(*p\_size: int = 1, p\_test\_data: float = 0.3, p\_batch\_size: int = 100, p\_seed: int = 1*)

Bases: Buffer, Dataset

This class provides buffer functionalities for PyTorch based SLNetwork and also using several built-in PyTorch functionalities.

### Parameters

- **p\_size** (int) – the buffer size. Default = 1.
- **p\_test\_data** (float) – the proportion of testing data within the sampled data. Default = 0.3.
- **p\_batch\_size** (int) – the batch size for a sample. Default = 100.
- **p\_seed** (int) – the seeding for randomizer in the buffer, optional. Default = 1.

**add\_element**(*p\_elem: BufferElement*)

This method has a functionality to add an element to the buffer.

### Parameters

**p\_elem** (BufferElement) – an element of the buffer

**get\_internal\_counter**() → int

This method has a functionality to get the number of elements being added to the buffer.

**sampling**()

This method has a functionality to sample from the buffer using built-in PyTorch functionalities.

### Returns

- **trainer** (*dict*) – a dictionary that consists of sampled data for training, which are splitted to 2 keys such as input and output. The value of each key is a torch's DataLoader of the sampled data.
- **tester** (*dict*) – a dictionary that consists of sampled data for testing, which are splitted to 2 keys such as input and output. The value of each key is a torch's DataLoader of the sampled data.

**class** mlpro.sl.pool.afct.pytorch.**PyTorchHelperFunctions**

Bases: object

PyTorch Helper Functions in MLPro-SL.

**input\_preproc**(*p\_input*: [Element](#)) → Tensor

This method has a functionality to transform input data in the form of Element to torch.Tensor for pre-processing.

**Parameters**

**p\_input** ([Element](#)) – Input data in the form of Element.

**Returns**

**input** – Input data in the form of torch.Tensor.

**Return type**

torch.Tensor

**output\_preproc**(*p\_output*: [Element](#)) → Tensor

This method has a functionality to transform output data in the form of Element to torch.Tensor for pre-processing.

**Parameters**

**p\_output** ([Element](#)) – Output data in the form of Element.

**Returns**

**output** – Output data in the form of torch.Tensor.

**Return type**

torch.Tensor

**output\_postproc**(*p\_output*: *Tensor*) → list

This method has a functionality to transform output data in the form of torch.Tensor to a list for post-processing.

**Parameters**

**p\_output** (*torch.Tensor*) – Output data in the form of torch.Tensor.

**Returns**

**output** – Output data in the form of list.

**Return type**

list

**\_input\_preproc**(*p\_input*: *Tensor*) → Tensor

Additional process of input\_preproc. This is optional. Please redefine if you need it.

**Parameters**

**p\_input** (*torch.Tensor*) – Input data in the form of torch.Tensor.

**Returns**

**input** – Processed input data in the form of torch.Tensor.

**Return type**

torch.Tensor

**\_output\_preproc**(*p\_output: Tensor*) → Tensor

Additional process of output\_preproc. This is optional. Please redefine if you need it.

**Parameters****p\_output** (*torch.Tensor*) – Output data in the form of torch.Tensor.**Returns****output** – Processed output data in the form of torch.Tensor.**Return type**

torch.Tensor

**\_output\_postproc**(*p\_output: Tensor*) → Tensor

Additional process of output\_postproc. This is optional. Please redefine if you need it.

**Parameters****p\_output** (*torch.Tensor*) – Output data in the form of torch.Tensor.**Returns****output** – Processed output data in the form of torch.Tensor.**Return type**

torch.Tensor

**\_default\_weight\_bias\_init**(*module, weight\_init, bias\_init, gain=1*)

Weight and bias initialization method.

## 11.2.3 MLPro-RL - Reinforcement Learning

### Action Planners

#### MPC - Model Predictive Control

Ver. 1.1.1 (2023-02-04)

This module provides a default implementation of model predictive control (MPC).

**class** mlpro.rl.pool.actionplanner.mpc.MPC(*p\_range\_max=0, p\_state\_thsld=1e-08, p\_logging=True*)Bases: [ActionPlanner](#), [ScientificObject](#), [Async](#)

Template class for MPC to be used as part of model-based planning agents. The goal is to find the best sequence of actions that leads to a maximum reward.

**Parameters**

- **p\_range** (*int*) – Range of asynchronicity.
- **p\_state\_thsld** (*float*) – Threshold for metric difference between two states to be equal. Default = 0.00000001.
- **p\_logging** – Log level (see constants of class Log). Default = Log.C\_LOG\_ALL.

**C\_TYPE** = 'Model Predictive Control'

**`_plan_action`**(*p\_obs*: *State*) → *SARSBuffer*

Custom planning algorithm to fill the internal action path (`self._action_path`). Search width and depth are restricted by the attributes `self._width_limit` and `self._prediction_horizon`. The default implementation utilizes MPC.

**Parameters**

**`p_obs`** (*State*) – Observation data.

**Returns**

**`action_path`** – Sequence of *SARSElement* objects with included actions that lead to the best possible reward.

**Return type**

*SARSBuffer*

**`execute`**(\*\**p\_kwargs*)

**`_async_subtask`**(*p\_tid*: *int*, *p\_obs*: *State*)

## Environments

### Bulk Goods Plant

Ver. 2.3.2 (2023-08-22)

This module provides an RL environment of Bulk Good Laboratory Plant (BGLP).

**`class mlpro.rl.pool.envs.bglp.Actuator`**(*minpower*, *maxpower*, *minaction*, *maxaction*, *masscoeff*)

Bases: *object*

This class serves as a parent class of different types of actuators, which provides the main attributes of an actuator in the BGLP environment.

**Parameters**

- **`minpower`** (*float*) – minimum power of an actuator.
- **`maxpower`** (*float*) – maximum power of an actuator.
- **`minaction`** (*float*) – minimum action of an actuator.
- **`maxaction`** (*float*) – maximum action of an actuator.
- **`masscoeff`** (*float*) – mass transport coefficient of an actuator.

**`reg_a`**

list of existing actuators in the environment.

**Type**

list of objects

**`idx_a`**

length of `reg_a`.

**Type**

*int*

**`power_max`**

maximum power of an actuator.

**Type**  
float

**power\_min**

minimum power of an actuator.

**Type**  
float

**power\_coeff**

power coefficient of an actuator, if necessary.

**Type**  
float

**action\_max**

maximum action of an actuator.

**Type**  
float

**action\_min**

minimum action of an actuator.

**Type**  
float

**mass\_coeff**

mass transport coefficient of an actuator.

**Type**  
float

**t\_activated**

a time indicator about an actuator is activated.

**Type**  
float

**t\_end**

a time indicator about the end of an activation sequence of the actuator.

**Type**  
float

**status**

status of an actuator, false means inactive and true means active.

**Type**  
bool

**cur\_mass\_transport**

current transported mass of an actuator.

**Type**  
float

**cur\_power**

current power consumption of an actuator.

**Type**  
float

**cur\_action**

current taken action of an actuator in RL context.

**Type**

float

**cur\_speed**

current speed of an actuator.

**Type**

float

**type\_**

a short name for an actuator, usually 3 capital letters (e.g VAC, BLT).

**Type**

str

**reg\_a** = []

**power\_coeff** = 0

**t\_activated** = 0

**t\_end** = 0

**status** = False

**type\_** = ''

**idx\_a** = 0

**power\_max** = 0

**power\_min** = 0

**action\_min** = 0

**action\_max** = 0

**mass\_coeff** = 0

**cur\_mass\_transport** = 0

**cur\_power** = 0

**cur\_action** = 0

**cur\_speed** = 0

```
class mlpro.rl.pool.envs.bglp.VacuumPump(name, minpower, maxpower, minaction, maxaction,
                                         masscoeff)
```

Bases: [Actuator](#)

This class inherits Actuator class and serves as a child class of Actuator. This class represents a type of actuators in the BGLP environment, namely Vacuum Pump. Vacuum Pumps are mostly used to transport material from mini hoppers to silos. However, the parameter of each vacuum pump can be dissimilar to each other based on their settings.

**Parameters**

- **name** (*str*) – specific name or id of a vacuum pump (e.g. Vac\_A, etc.).

- **minpower** (*float*) – minimum power of a vacuum pump.
- **maxpower** (*float*) – maximum power of a vacuum pump.
- **minaction** (*float*) – minimum action of a vacuum pump.
- **maxaction** (*float*) – maximum action of a vacuum pump.
- **masscoeff** (*float*) – mass transport coefficient of a vacuum pump.

**reg\_v**

list of existing vacuum pumps.

**Type**

list of objects

**idx\_v**

length of reg\_v.

**Type**

int

**name**

specific name or id of a vacuum pump.

**Type**

str

**t\_end\_max**

maximum end of activation time of a vacuum pump with respect to current time.

**Type**

float

**reg\_v** = []

**t\_end\_max** = 0

**idx\_v** = 0

**name** = ''

**start\_t**(*now, duration, overwrite=False*)

This method calculates the activation time and the end of activation time of the vacuum pump. This method is called, if the vacuum pump would like to be activated or updated.

**Parameters**

- **now** (*float*) – current time of the system.
- **duration** (*float*) – duration of the vacuum pump being activated or the action by an agent in RL context.
- **overwrite** (*bool, optional*) – To indicate whether the current operation can be overwritten or not.

**calc\_mass**(*now*)

This method calculates the transported mass flow by the vacuum pump for a time step.

**Parameters**

**now** (*float*) – current time of the system.

**Returns**

**cur\_mass\_transport** – current transported mass.

**Return type**

float

**calc\_power()**

This method calculates the power consumption of a vacuum pump.

**Returns**

**cur\_power** – current power consumption.

**Return type**

float

**update(now)**

This method calculates whether a vacuum pump must be deactivated or not.

**Parameters**

**now** (*float*) – current time of the system.

**deactivate()**

This method is used to deactivate a vacuum pump.

```
class mlpro.rl.pool.envs.bglp.Belt(name, actiontype, minpower, maxpower, minaction, maxaction,
                                   masscoeff)
```

Bases: [Actuator](#)

This class inherits Actuator class and serves as a child class of Actuator. This class represents a type of actuators in the BGLP environment, namely Belt. This class can be used for Conveyor Belt, Rotary Feeder, Vibratory Conveyor, or similar type of actuators. Belts are mostly used to transport material from silos to hoppers. However, the parameter of each actuator can be dissimilar to each other based on their settings.

**Parameters**

- **name** (*str*) – specific name or id of an actuator (e.g. Belt\_A, etc.).
- **actiontype** (*str*) – “C” for continuous action, “B” for binary action.
- **minpower** (*float*) – minimum power of an actuator.
- **maxpower** (*float*) – maximum power of an actuator.
- **minaction** (*float*) – minimum action of an actuator.
- **maxaction** (*float*) – maximum action of an actuator.
- **masscoeff** (*float*) – mass transport coefficient of an actuator.

**reg\_b**

list of existing actuators.

**Type**

list of objects

**idx\_b**

length of reg\_b.

**Type**

int

**name**

specific name or id of an actuator.

**Type**

str



**actiontype**

“C” for continuous action, “B” for binary action.

**Type**

str

**speed**

speed of an actuator.

**Type**

float

**reg\_b** = []

**idx\_b** = 0

**name** = ''

**actiontype** = ''

**speed** = 0

**start\_t**(*now, duration, speed, overwrite=False*)

This method calculates the activation time and the end of activation time of the belt. This method is called, if the belt would like to be activated or updated.

**Parameters**

- **now** (*float*) – current time of the system.
- **duration** (*float*) – duration of the belt being activated, which being defined by the time set.
- **speed** (*float*) – speed of the belt or the action by an agent in RL context.
- **overwrite** (*bool, optional*) – To indicate whether the current operation can be overwritten or not.

**calc\_mass**(*now*)

This method calculates the transported mass flow by the belt for a time step.

**Parameters**

**now** (*float*) – current time of the system.

**Returns**

**cur\_mass\_transport** – current transported mass.

**Return type**

float

**calc\_power**()

This method calculates the power consumption of a belt.

**Returns**

**cur\_power** – current power consumption.

**Return type**

float

**update**(*now*)

This method calculates whether a belt must be deactivated or not.

**Parameters**

**now** (*float*) – current time of the system.

**deactivate()**

This method is used to deactivate a belt.

**class** mlpro.rl.pool.envs.bglp.**Reservoir**(*vol\_max*, *vol\_init\_abs=0*)

Bases: object

This class serves as a parent class of different types of reservoirs, which provides the main attributes of a buffer in the BGLP environment.

**Parameters**

- **vol\_max** (*float*) – maximum volume of a reservoir.
- **vol\_init\_abs** (*float*, *optional*) – initial volume of a reservoir. The default is 0.

**reg\_r**

list of existing reservoirs in the environment.

**Type**

list of objects

**idx\_r**

length of reg\_r.

**Type**

int

**vol\_max**

maximum volume of a reservoir.

**Type**

float

**vol\_init\_abs**

initial volume of a reservoir.

**Type**

float

**vol\_cur\_abs**

current volume of a reservoir.

**Type**

float

**vol\_cur\_rel**

current volume of a reservoir in percentage.

**Type**

float

**change**

volume change of a reservoir in a time step.

**Type**

float

**reg\_r** = []

```

idx_r = []
vol_max = 0
vol_init_abs = 0
vol_cur_abs = 0
vol_cur_rel = 0
change = 0

```

**set\_change**(*vol\_change*)

This method sets up a volume change of a reservoir.

**Parameters**

**vol\_change** (*float*) – volume change of a reservoir in a time step.

**update**()

This method calculates the current volume of a reservoir after volume changes are made.

**class** mlpro.rl.pool.envs.bglp.**Silo**(*name*, *vol\_max*, *vol\_cur*=0, *mode*='abs')

Bases: [Reservoir](#)

This class inherits Reservoir class and serves as a child class of Reservoir. This class represents a type of buffers in the BGLP environment, namely Silo. Silos are used to temporarily store the transported materials. However, the parameter of each silo can be dissimilar to each other based on their settings.

**Parameters**

- **name** (*str*) – specific name or id of a silo (e.g. Silo\_A, etc.).
- **vol\_max** (*float*) – maximum capacity of a silo.
- **vol\_cur** (*float*) – current volume of a silo.
- **mode** (*str*, *optional*) – mode of measuring the current volume. “abs” means absolute value, “rel” means percentage. The default is “abs”.

**reg\_s**

list of existing silos.

**Type**

list of objects

**idx\_s**

length of reg\_s.

**Type**

int

**name**

specific name or id of a silo.

**Type**

str

**type\_**

a short name for a silo, usually 3 capital letters (e.g. SIL).

**Type**

str

```
reg_s = []  
name = ''  
idx_s = []  
type_ = ''
```

```
class mlpro.rl.pool.envs.bglp.Hopper(name, vol_max, vol_cur=0, mode='abs')
```

Bases: [Reservoir](#)

This class inherits Reservoir class and serves as a child class of Reservoir. This class represents a type of buffers in the BGLP environment, namely Hopper. Hoppers are used to temporary stored the transported materials. However, the parameter of each hopper can be dissimilar to each other based on their settings.

#### Parameters

- **name** (*str*) – specific name or id of a hopper (e.g. Hop\_A, etc.).
- **vol\_max** (*float*) – maximum capacity of a hopper.
- **vol\_cur** (*float*) – current volume of a hopper.
- **mode** (*str, optional*) – mode of measuring the current volume. “abs” means absolute value, “rel” means percentage. The default is “abs”.

**reg\_h**

list of existing silos.

**Type**

list of objects

**idx\_h**

length of reg\_h.

**Type**

int

**name**

specific name or id of a hopper.

**Type**

str

**type\_**

a short name for a hopper, usually 3 capital letters (e.g HOP).

**Type**

str

```
reg_h = []
```

```
name = ''
```

```
idx_h = []
```

```
type_ = ''
```

```
class mlpro.rl.pool.envs.bglp.BGLP(p_reward_type=1, p_visualize: bool = False, p_logging=True,  
    t_step=0.5, t_set=10.0, demand=0.1, lr_margin=1.0, lr_demand=4.0,  
    lr_power=0.001, margin_p=[0.2, 0.8, 4], prod_target=10000,  
    prod_scenario='continuous', cycle_limit=0)
```

Bases: [Environment](#)

This is the main class of BGLP environment that inherits Environment class from MLPro.

#### Parameters

- **p\_reward\_type** ([Reward](#), *optional*) – rewarding type. The default is Reward.C\_TYPE\_EVERY\_AGENT.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_logging** ([Log](#), *optional*) – logging functionalities. The default is Log.C\_LOG\_ALL.
- **t\_step** (*float*, *optional*) – time for each time step (in seconds). The default is 0.5.
- **t\_set** (*float*, *optional*) – time set for one horizon (in seconds). The default is 10.0.
- **demand** (*float*, *optional*) – the constant output flow from the system (in L/s). The default is 0.1.
- **lr\_margin** (*float*, *optional*) – the learning rate for margin parameter, related to implemented reward function. The default is 1.0.
- **lr\_demand** (*float*, *optional*) – the learning rate for production demand, related to implemented reward function. The default is 4.0.
- **lr\_power** (*float*, *optional*) – the learning rate for power consumption, related to implemented reward function. The default is 0.0010.
- **margin\_p** (*list of floats*, *optional*) – the margin parameter of reservoirs [low, high, multiplier]. The default is [0.2,0.8,4].
- **prod\_target** (*float*, *optional*) – the production target for batch operation (in L). The default is 10000.
- **prod\_scenario** (*str*, *optional*) – ‘batch’ means batch production scenario and ‘continuous’ means continuous production scenario. The default is ‘continuous’.

#### C\_NAME

name of the environment.

##### Type

str

#### C\_LATENCY

latency.

##### Type

timedelta()

#### C\_INFINITY

infinity.

##### Type

np.finfo()

#### C\_REWARD\_TYPE

rewarding type.

##### Type

[Reward](#)

**sil**

list of existing silos.

**Type**

list of objects

**hops**

list of existing hoppers.

**Type**

list of objects

**ress**

list of existing reservoirs.

**Type**

list of objects

**blts**

list of existing belts.

**Type**

list of objects

**acts**

list of existing actuators.

**Type**

list of objects

**vacs**

list of existing vacuum pumps.

**Type**

list of objects

**con\_act\_to\_res**

connection between actuators and reservoirs and setup the sequence.

**Type**

list of lists of int

**m\_param**

the margin parameter of reservoirs [low, high, multiplicator].

**Type**

list of floats

**\_demand**

the constant output flow from the system (in L/s).

**Type**

float

**t**

current time of the system (in seconds).

**Type**

float

**t\_step**

time for each time step (in seconds).

**Type**

float

**lr\_margin**

the learning rate for margin parameter.

**Type**

float

**lr\_demand**

the learning rate for production demand.

**Type**

float

**lr\_power**

the learning rate for power consumption.

**Type**

float

**overflow**

current overflow.

**Type**

list of floats

**power**

current power consumption.

**Type**

list of floats

**transport**

current transported mass flow.

**Type**

list of floats

**reward**

current rewards.

**Type**

list of floats

**levels\_init**

initial level of reservoirs.

**Type**

float

**overflow\_t**

current overflow for a specific buffer in a specific time.

**Type**

float

**demand\_t**

current demand for a specific buffer in a specific time.

**Type**

float

**power\_t**

current power consumption for a specific actuator in a specific time.

**Type**

float

**transport\_t**

current transported material by a specific actuator in a specific time.

**Type**

float

**margin\_t**

current margin for a specific buffer in a specific time.

**Type**

float

**prod\_reached**

current production reached in L for batch operation.

**Type**

float

**prod\_target**

the production target for batch operation (in L).

**Type**

float

**prod\_scenario**

'batch' means batch production scenario and 'continuous' means continuous production scenario.

**Type**

str

**cycle\_limit**

the number of cycle limit.

**Type**

int

**C\_NAME** = 'BGLP'

**C\_LATENCY** = `datetime.timedelta(seconds=1)`

**C\_INFINITY** = 3.4028235e+38

**C\_REWARD\_TYPE** = 1

**con\_act\_to\_res** = []

**m\_param** = []

**prod\_reached** = 0



```

C_CYCLE_LIMIT = 0

t = 0

t_step = 0

_demand = 0

lr_margin = 0

lr_demand = 0

lr_power = 0

prod_target = 0

prod_scenario = 0

levels_init = 0

sils = []

hops = []

ress = []

blts = []

vacs = []

acts = []

overflow = []

demand = []

power = []

transport = []

overflow_t = 0

demand_t = 0

power_t = 0

transport_t = 0

margin_t = 0

reward = []

```

#### **static setup\_spaces()**

This method is used to setup action and state spaces of the system.

The actions and states are normalized between 0 to 1. For the actions, 0 means minimum action and 1 means maximum action. Meanwhile, for the states, 0 means minimum capacity (empty) and 1 means maximum capacity (full)

#### **Returns**

- **state\_space** (*ESpace()*) – state space of the system.

- **action\_space** (*ESpace()*) – action space of the system.

**collect\_substates()** → *State*

This method is called during resetting the environment.

**Returns**

**state** – current states.

**Return type**

*State*

**\_reset**(*p\_seed=None*) → None

This method is used to reset the environment.

**Parameters**

**p\_seed** (*int*) – Seed parameter for an internal random generator.

**\_simulate\_reaction**(*p\_state: State, p\_action: Action*) → *State*

This method is used to calculate the next states of the system after a set of actions.

**Parameters**

- **p\_state** (*State*) – State.
- **p\_action** (*Action*) – Action.

**Returns**

**\_state** – current states.

**Return type**

*State*

**\_compute\_success**(*p\_state: State*) → bool

This method computes the success flag. This method can be redefined.

**Parameters**

**p\_state** (*State*) – state.

**Returns**

success or not success.

**Return type**

bool

**\_compute\_broken**(*p\_state: State*) → bool

This method computes the broken flag. This method can be redefined.

**Parameters**

**p\_state** (*State*) – state.

**Returns**

broken or not.

**Return type**

bool

**\_compute\_reward**(*p\_state\_old: State, p\_state\_new: State*) → *Reward*

This method calculates the reward for different reward types.

**Parameters**

- **p\_state\_old** (*State*) – previous state.
- **p\_state\_new** (*State*) – new state.

**Returns**

**reward** – reward values.

**Return type**

*Reward*

**calc\_mass\_flows()**

This method calculates the mass flow transported by actuators.

**calc\_power()**

This method calculates the power consumptions per actuator.

**calc\_margin()**

This method calculates margin level for every reservoir.

**set\_volume\_changes(*demandval*)**

This method sets up volume changes for every reservoir.

**update\_levels()**

This method updates the current level of reservoirs.

**reset\_levels()**

This method resets reservoirs.

**reset\_actuators()**

This method resets actuators.

**update\_actuators()**

This method updates actuators.

**update(*demandval*)**

This method sets up volume changes, updates reservoirs' level, and updates actuators.

**Parameters**

**demandval** (*float*) – production outflow target in L/s.

**get\_status(*t*, *demandval*)**

This method calculates overflow, demand, power, transport, and margin. This function will be called every time step.

**Parameters**

- **t** (*float*) – current time in sec.
- **demandval** (*float*) – production outflow target in L/s.

**Returns**

- **overflow** (*list of floats*) – overflow levels.
- **demand** (*list of floats*) – demand fulfilled.
- **power** (*list of floats*) – power consumptions.
- **transport** (*list of floats*) – transported materials.
- **margin** (*list of floats*) – margin levels.

**set\_actions(*actions*)**

This method sets up actions for actuators. This function will be called every time set.

**calc\_state()**

This method obtains current levels of reservoirs.

**Returns**

**levels** – level of each reservoir.

**Return type**

list of floats

**calc\_reward()**

This method calculates the reward. This method can be redefined! The current reward function is suitable for continuous operation and scalar reward for individual agents.

**Returns**

**reward** – reward for each agent.

**Return type**

list of floats

**init\_plot(*p\_figure=None*)**

Initializes the plot functionalities of the class.

**Parameters**

- **p\_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p\_plot\_settings** (*PlotSettings*) – Optional plot settings. If None, the default view is plotted (see attribute C\_PLOT\_DEFAULT\_VIEW).

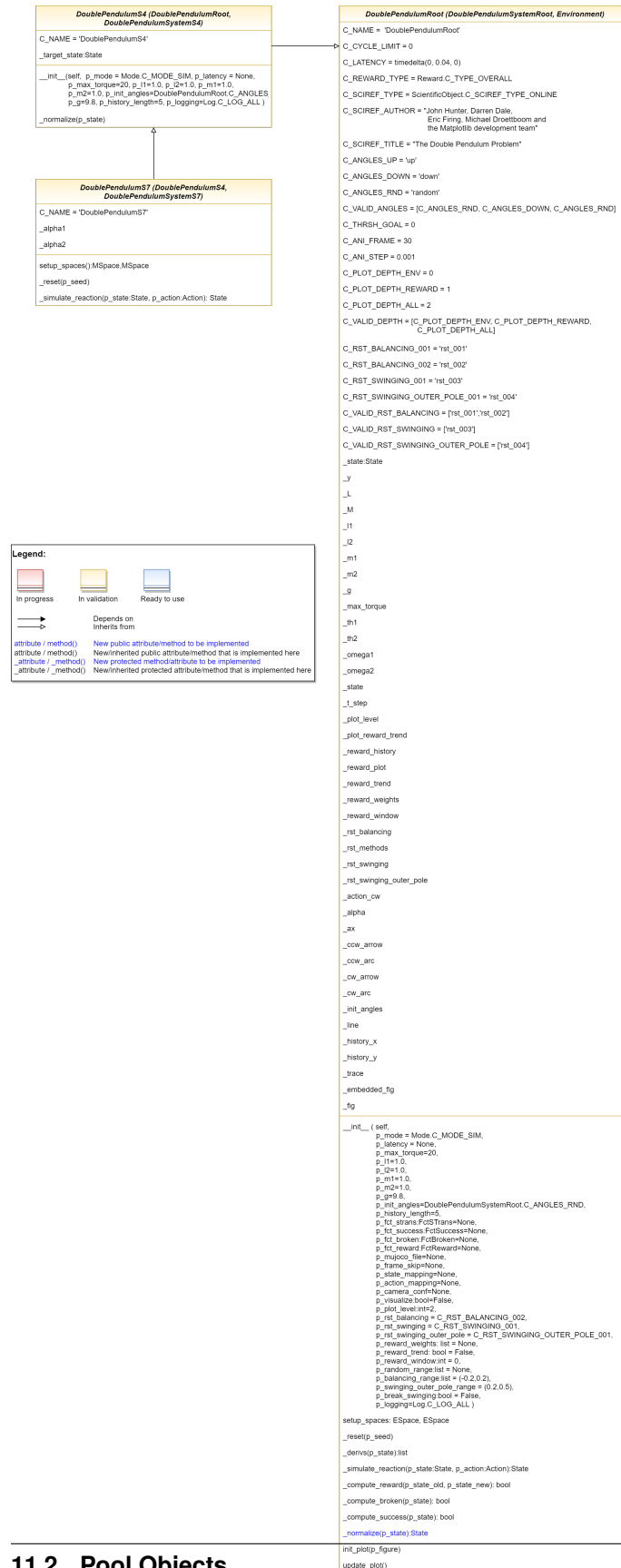
**update\_plot()**

Updates the plot.

**Parameters**

**\*\*p\_kwargs** – Implementation-specific plot data and/or parameters.

## Double Pendulum



The Double Pendulum environment is an implementation of a classic control problem of Double Pendulum system. The dynamics of the system are based on the [Double Pendulum](#) implementation by [Matplotlib](#). The double pendulum is a system of two poles, with the inner pole connected to a fixed point at one end and to outer pole at other end. The native implementation of Double Pendulum consists of an input motor providing the torque in either directions to actuate the system.

```
class mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot(p_id=None, p_name=None,  
                                                           p_buffer_size=0, p_range_max=0,  
                                                           p_autorun=0, p_class_shared=None,  
                                                           p_mode=0, p_latency=None,  
                                                           p_max_torque=20, p_l1=1.0,  
                                                           p_l2=1.0, p_m1=1.0, p_m2=1.0,  
                                                           p_g=9.8, p_init_angles='random',  
                                                           p_history_length=5, p_fct_strans:  
                                                           FctSTrans = None, p_fct_success:  
                                                           FctSuccess = None, p_fct_broken:  
                                                           FctBroken = None, p_fct_reward:  
                                                           FctReward = None,  
                                                           p_mujoco_file=None,  
                                                           p_frame_skip=None,  
                                                           p_state_mapping=None,  
                                                           p_action_mapping=None,  
                                                           p_camera_conf=None, p_visualize:  
                                                           bool = False, p_plot_level: int = 2,  
                                                           p_rst_balancing='rst_002',  
                                                           p_rst_swinging='rst_003',  
                                                           p_rst_swinging_outer_pole='rst_004',  
                                                           p_reward_weights: list = None,  
                                                           p_reward_trend: bool = False,  
                                                           p_reward_window: int = 0,  
                                                           p_random_range: list = None,  
                                                           p_balancing_range: list = (-0.2, 0.2),  
                                                           p_swinging_outer_pole_range=(0.2,  
                                                           0.5), p_break_swinging: bool = False,  
                                                           p_logging=True)
```

Bases: [DoublePendulumSystemRoot](#), [Environment](#)

This is the root double pendulum environment class inherited from Environment class with four dimensional state space and underlying implementation of the Double Pendulum dynamics, default reward strategy.

#### Parameters

- **p\_mode** – Mode of environment. Possible values are Mode.C\_MODE\_SIM(default) or Mode.C\_MODE\_REAL.
- **p\_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C\_LATENCY is used by default.
- **p\_max\_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p\_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p\_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p\_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p\_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25

- **p\_g** (*float, optional*) – Gravitational acceleration. The default is 9.8
- **p\_init\_angles** (*str, optional*) – C\_ANGLES\_UP starts the pendulum in an upright position C\_ANGLES\_DOWN starts the pendulum in a downward position C\_ANGLES\_RND starts the pendulum from a random position.
- **p\_history\_length** (*int, optional*) – Historical trajectory points to display. The default is 5.
- **p\_fct\_strans** (*FctSTrans, optional*) – The custom State transition function.
- **p\_fct\_success** (*FctSuccess, optional*) – The custom Success Function.
- **p\_fct\_broken** (*FctBroken, optional*) – The custom Broken Function.
- **p\_fct\_reward** (*FctReward, optional*) – The custom Reward Function.
- **p\_mujoco\_file** (*optional*) – The corresponding mujoco file
- **p\_frame\_skip** (*optional*) – Number of frames to be skipped for visualization.
- **p\_state\_mapping** (*optional*) – State mapping configurations.
- **p\_action\_mapping** (*optional*) – Action mapping configurations.
- **p\_camera\_conf** (*optional*) – Camera configurations for mujoco specific visualization.
- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_plot\_level** (*int*) – Types and number of plots to be plotted. Default = ALL C\_PLOT\_DEPTH\_ENV only plots the environment C\_PLOT\_DEPTH\_REWARD only plots the reward C\_PLOT\_ALL plots both reward and the environment
- **p\_rst\_balancing** – Reward strategy to be used for the balancing region of the environment
- **p\_rst\_swinging** – Reward strategy to be used for the swinging region of the environment
- **p\_rst\_swinging\_outer\_pole** – Reward Strategy to be used for swinging up outer pole
- **p\_reward\_weights** (*list*) – List of weights to be added to the dimensions of the state space for reward computation
- **p\_reward\_trend** (*bool*) – Boolean value stating whether to plot reward trend
- **p\_reward\_window** (*int*) – The number of latest rewards to be shown in the plot. Default is 0
- **p\_random\_range** (*list*) – The boundaries for state space for initialization of environment randomly
- **range** (*p\_balancing*) – The boundaries for state space of environment in balancing region
- **p\_swinging\_outer\_pole\_range** – The boundaries for state space of environment in swinging of outer pole region
- **p\_break\_swinging** (*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p\_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C\_LOG\_WE.

**C\_REWARD\_TYPE** = 0

**C\_PLOT\_DEPTH\_ENV** = 0

**C\_PLOT\_DEPTH\_REWARD** = 1

```
C_PLOT_DEPTH_ALL = 2
C_VALID_DEPTH = [0, 1, 2]
C_RST_BALANCING_001 = 'rst_001'
C_RST_BALANCING_002 = 'rst_002'
C_RST_SWINGING_001 = 'rst_003'
C_RST_SWINGING_OUTER_POLE_001 = 'rst_004'
C_VALID_RST_BALANCING = ['rst_001', 'rst_002']
C_VALID_RST_SWINGING = ['rst_003']
C_VALID_RST_SWINGING_OUTER_POLE = ['rst_004']
```

**`_compute_reward`**(*p\_state\_old*: *State = None*, *p\_state\_new*: *State = None*) → *Reward*

This method calculates the reward for C\_TYPE\_OVERALL reward type. The current reward is based on the worst possible distance between two states and the best possible minimum distance between current and goal state.

**Parameters**

- **`p_state_old`** (*State*) – Previous state.
- **`p_state_new`** (*State*) – New state.

**Returns**

**`current_reward`** – current calculated Reward values.

**Return type**

*Reward*

**`compute_reward_001`**(*p\_state\_old*: *State = None*, *p\_state\_new*: *State = None*)

Reward strategy with only new normalized state

**Parameters**

- **`p_state_old`** (*State*) – Normalized old state.
- **`p_state_new`** (*State*) – Normalized new state.

**Returns**

**`current_reward`** – current calculated Reward values.

**Return type**

*Reward*

**`compute_reward_002`**(*p\_state\_old*: *State = None*, *p\_state\_new*: *State = None*)

Reward strategy with both new and old normalized state based on euclidean distance from the goal state. Designed the balancing zone.

**Parameters**

- **`p_state_old`** (*State*) – Normalized old state.
- **`p_state_new`** (*State*) – Normalized new state.

**Returns**

**`current_reward`** – current calculated Reward values.



**Return type***Reward***compute\_reward\_003**(*p\_state\_old*: *State* = None, *p\_state\_new*: *State* = None)

Reward strategy with both new and old normalized state based on euclidean distance from the goal state, designed for the swinging of outer pole. Both angles and velocity and acceleration of the outer pole are considered for the reward computation.

**Parameters**

- **p\_state\_old** (*State*) – Normalized old state.
- **p\_state\_new** (*State*) – Normalized new state.

**Returns****current\_reward** – current calculated Reward values.**Return type***Reward***compute\_reward\_004**(*p\_state\_old*: *State* = None, *p\_state\_new*: *State* = None)

Reward strategy with both new and old normalized state based on euclidean distance from the goal state, designed for the swinging up region. Both angles and velocity and acceleration of the outer pole are considered for the reward computation.

The reward strategy is as follows:

$$\text{reward} = (|\text{old\_theta}| - |\text{new\_theta}|) + (|\text{new\_omega} + \text{new\_alpha}| - |\text{old\_omega} + \text{old\_alpha}|)$$
**Parameters**

- **p\_state\_old** (*State*) – Normalized old state.
- **p\_state\_new** (*State*) – Normalized new state.

**Returns****current\_reward** – current calculated Reward values.**Return type***Reward***\_init\_plot\_2d**(*p\_figure*: *Figure*, *p\_settings*: *PlotSettings*)

Custom method to initialize a 2D plot. If attribute *p\_settings.axes* is not None the initialization shall be done there. Otherwise a new Matplotlib Axes object shall be created in the given figure and stored in *p\_settings.axes*.

**Parameters**

- **p\_figure** (*Matplotlib.figure.Figure*) – Matplotlib figure object to host the sub-plot(s).
- **p\_settings** (*PlotSettings*) – Object with further plot settings.

**\_update\_plot\_2d**(*p\_settings*: *PlotSettings*, *\*\*p\_kwargs*)

This method updates the plot figure of each episode. When the figure is detected to be an embedded figure, this method will only set up the necessary data of the figure.

```
class mlpro.rl.pool.envs.doublependulum.DoublePendulumS4(p_id=None, p_name=None,
                                                           p_buffer_size=0, p_range_max=0,
                                                           p_autorun=0, p_class_shared=None,
                                                           p_mode=0, p_latency=None,
                                                           p_max_torque=20, p_l1=1.0, p_l2=1.0,
                                                           p_m1=1.0, p_m2=1.0, p_g=9.8,
                                                           p_init_angles='random',
                                                           p_history_length=5, p_fct_strans:
                                                           FctSTrans = None, p_fct_success:
                                                           FctSuccess = None, p_fct_broken:
                                                           FctBroken = None, p_fct_reward:
                                                           FctReward = None, p_mujoco_file=None,
                                                           p_frame_skip=None,
                                                           p_state_mapping=None,
                                                           p_action_mapping=None,
                                                           p_camera_conf=None, p_visualize: bool
                                                           = False, p_plot_level: int = 2,
                                                           p_rst_balancing='rst_002',
                                                           p_rst_swinging='rst_003',
                                                           p_rst_swinging_outer_pole='rst_004',
                                                           p_reward_weights: list = None,
                                                           p_reward_trend: bool = False,
                                                           p_reward_window: int = 0,
                                                           p_random_range: list = None,
                                                           p_balancing_range: list = (-0.2, 0.2),
                                                           p_swinging_outer_pole_range=(0.2, 0.5),
                                                           p_break_swinging: bool = False,
                                                           p_logging=True)
```

Bases: [DoublePendulumRoot](#), [DoublePendulumSystemS4](#)

This is the Double Pendulum Static 4 dimensional environment that inherits from the double pendulum root class, inheriting the dynamics and default reward strategy.

#### Parameters

- **p\_mode** – Mode of environment. Possible values are Mode.C\_MODE\_SIM(default) or Mode.C\_MODE\_REAL.
- **p\_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C\_LATENCY is used by default.
- **p\_max\_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p\_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p\_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p\_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p\_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25
- **p\_init\_angles** (*str, optional*) – C\_ANGLES\_UP starts the pendulum in an upright position C\_ANGLES\_DOWN starts the pendulum in a downward position C\_ANGLES\_RND starts the pendulum from a random position.
- **p\_g** (*float, optional*) – Gravitational acceleration. The default is 9.8
- **p\_history\_length** (*int, optional*) – Historical trajectory points to display. The default is 5.

- **p\_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p\_plot\_level** (*int*) – Types and number of plots to be plotted. Default = ALL  
C\_PLOT\_DEPTH\_ENV only plots the environment C\_PLOT\_DEPTH\_REWARD only plots the reward C\_PLOT\_ALL plots both reward and the environment
- **p\_rst\_balancingL** – Reward strategy to be used for the balancing region of the environment
- **p\_rst\_swinging** – Reward strategy to be used for the swinging region of the environment
- **p\_reward\_weights** (*list*) – List of weights to be added to the dimensions of the state space for reward computation
- **p\_reward\_trend** (*bool*) – Boolean value stating whether to plot reward trend
- **p\_reward\_window** (*int*) – The number of latest rewards to be shown in the plot. Default is 0
- **p\_random\_range** (*list*) – The boundaries for state space for initialization of environment randomly
- **range** (*p\_balancing*) – The boundaries for state space of environment in balancing region
- **p\_break\_swinging** (*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p\_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C\_LOG\_WE.

**C\_NAME** = 'DoublePendulumS4'

**\_normalize**(*p\_state: list*)

Method for normalizing the State values of the DP env based on MinMax normalisation based on static boundaries provided by MLPro.

**Parameters**

**p\_state** – The state to be normalized

**Returns**

Normalized state values

**Return type**

state

**\_obs\_to\_mujoco**(*p\_state*)

```
class mlpro.rl.pool.envs.doublependulum.DoublePendulumS7(p_id=None, p_name=None,
    p_buffer_size=0, p_range_max=0,
    p_autorun=0, p_class_shared=None,
    p_mode=0, p_latency=None,
    p_max_torque=20, p_l1=1.0, p_l2=1.0,
    p_m1=1.0, p_m2=1.0, p_g=9.8,
    p_init_angles='random',
    p_history_length=5, p_fct_strans:
    FctSTrans = None, p_fct_success:
    FctSuccess = None, p_fct_broken:
    FctBroken = None, p_fct_reward:
    FctReward = None, p_mujoco_file=None,
    p_frame_skip=None,
    p_state_mapping=None,
    p_action_mapping=None,
    p_camera_conf=None, p_visualize: bool
    = False, p_plot_level: int = 2,
    p_rst_balancing='rst_002',
    p_rst_swinging='rst_003',
    p_rst_swinging_outer_pole='rst_004',
    p_reward_weights: list = None,
    p_reward_trend: bool = False,
    p_reward_window: int = 0,
    p_random_range: list = None,
    p_balancing_range: list = (-0.2, 0.2),
    p_swinging_outer_pole_range=(0.2, 0.5),
    p_break_swinging: bool = False,
    p_logging=True)
```

Bases: [DoublePendulumS4](#), [DoublePendulumSystemS7](#)

This is the classic implementation of Double Pendulum with 7 dimensional state space including derived accelerations of both the poles and the input torque. The dynamics of the system are inherited from the Double Pendulum Root class.

#### Parameters

- **p\_mode** – Mode of environment. Possible values are Mode.C\_MODE\_SIM(default) or Mode.C\_MODE\_REAL.
- **p\_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C\_LATENCY is used by default.
- **p\_max\_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p\_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p\_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p\_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p\_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25
- **p\_init\_angles** (*str, optional*) – C\_ANGLES\_UP starts the pendulum in an upright position C\_ANGLES\_DOWN starts the pendulum in a downward position C\_ANGLES\_RND starts the pendulum from a random position.
- **p\_g** (*float, optional*) – Gravitational acceleration. The default is 9.8

- **p\_history\_length**(*int*, *optional*) – Historical trajectory points to display. The default is 5.
- **p\_visualize**(*bool*) – Boolean switch for visualisation. Default = False.
- **p\_plot\_level**(*int*) – Types and number of plots to be plotted. Default = ALL  
C\_PLOT\_DEPTH\_ENV only plots the environment C\_PLOT\_DEPTH\_REWARD only plots the reward C\_PLOT\_ALL plots both reward and the environment
- **p\_rst\_balancingL** – Reward strategy to be used for the balancing region of the environment
- **p\_rst\_swinging** – Reward strategy to be used for the swinging region of the environment
- **p\_reward\_weights**(*list*) – List of weights to be added to the dimensions of the state space for reward computation
- **p\_reward\_trend**(*bool*) – Boolean value stating whether to plot reward trend
- **p\_reward\_window**(*int*) – The number of latest rewards to be shown in the plot. Default is 0
- **p\_random\_range**(*list*) – The boundaries for state space for initialization of environment randomly
- **range**(*p\_balancing*) – The boundaries for state space of environment in balancing region
- **p\_break\_swinging**(*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p\_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C\_LOG\_WE.

**C\_NAME** = 'DoublePendulumS7'

```
class mlpro.rl.pool.envs.doublependulum.DoublePendulumOA4(p_id=None, p_name: str = None,
    p_buffer_size: int = 0, p_range_max: int
    = 0, p_autorun: int = 0, p_class_shared:
    Shared = None, p_mode=0,
    p_adapt=True, p_latency=None, p_t_step:
    timedelta = None, p_max_torque=20,
    p_l1=1.0, p_l2=1.0, p_m1=1.0,
    p_m2=1.0, p_g=9.8,
    p_init_angles='random',
    p_history_length=5, p_fct_strans:
    FctSTrans = None, p_fct_success:
    FctSuccess = None, p_fct_broken:
    FctBroken = None, p_fct_reward:
    FctReward = None, p_wf: OAWorkflow
    = None, p_wf_reward: OAWorkflow =
    None, p_wf_success: OAWorkflow =
    None, p_wf_broken: OAWorkflow =
    None, p_plot_level: int = 2,
    p_rst_balancing='rst_002',
    p_rst_swinging='rst_003',
    p_rst_swinging_outer_pole='rst_004',
    p_reward_weights: list = None,
    p_reward_trend: bool = False,
    p_reward_window: int = 0,
    p_random_range: list = None,
    p_balancing_range: list = (-0.2, 0.2),
    p_swinging_outer_pole_range=(0.2,
    0.5), p_break_swinging: bool = False,
    p_mujoco_file=None, p_frame_skip: int
    = 1, p_state_mapping=None,
    p_action_mapping=None,
    p_camera_conf: tuple = (None, None,
    None), p_visualize: bool = False,
    p_logging: bool = True, **p_kwargs)
```

Bases: OAEnvironment, [DoublePendulumS4](#), DoublePendulumOA4

C\_NAME = 'Double Pendulum A4'

```

class mlpro.rl.pool.envs.doublependulum.DoublePendulumOA7(p_id=None, p_name: str = None,
    p_buffer_size: int = 0, p_range_max: int
    = 0, p_autorun: int = 0, p_class_shared:
    Shared = None, p_mode=0,
    p_adapt=True, p_latency=None, p_t_step:
    timedelta = None, p_max_torque=20,
    p_l1=1.0, p_l2=1.0, p_m1=1.0,
    p_m2=1.0, p_g=9.8,
    p_init_angles='random',
    p_history_length=5, p_fct_strans:
    FctSTrans = None, p_fct_success:
    FctSuccess = None, p_fct_broken:
    FctBroken = None, p_fct_reward:
    FctReward = None, p_wf: OAWorkflow
    = None, p_wf_reward: OAWorkflow =
    None, p_wf_success: OAWorkflow =
    None, p_wf_broken: OAWorkflow =
    None, p_plot_level: int = 2,
    p_rst_balancing='rst_002',
    p_rst_swinging='rst_003',
    p_rst_swinging_outer_pole='rst_004',
    p_reward_weights: list = None,
    p_reward_trend: bool = False,
    p_reward_window: int = 0,
    p_random_range: list = None,
    p_balancing_range: list = (-0.2, 0.2),
    p_swinging_outer_pole_range=(0.2,
    0.5), p_break_swinging: bool = False,
    p_mujoco_file=None, p_frame_skip: int
    = 1, p_state_mapping=None,
    p_action_mapping=None,
    p_camera_conf: tuple = (None, None,
    None), p_visualize: bool = False,
    p_logging: bool = True, **p_kwargs)

```

Bases: [DoublePendulumOA4](#), [DoublePendulumOA7](#)

**C\_NAME** = 'Double Pendulum A7'

**C\_PLOT\_ACTIVE**: bool = True

## Grid World

Ver. 2.0.4 (2023-04-12)

This module provides an environment of customizable Gridworld.

```

class mlpro.rl.pool.envs.gridworld.GridWorld(p_logging: bool = True, p_grid_size=(8, 8),
    p_random_start_position: bool = True,
    p_random_goal_position: bool = True, p_max_step: int
    = 50, p_action_type: int = 0, p_visualize=True,
    p_start_position=None, p_goal_position=None)

```

Bases: [Environment](#)

Custom environment of an n-D grid world where the agent has to go to a random or defined target.

**Parameters**

- **p\_logging** (*bool*) – Subspace of an environment that is observed by the policy. Default = Log.C\_LOG\_ALL.
- **p\_grid\_size** (*dimension*) – Dimension of the grid world (n-D grid world), e.g. (8,8) for 2-D or (8,8,8) for 3-D. Default = (8,8)
- **p\_random\_start\_position** (*bool*) – Randomize start position. Default = True.
- **p\_random\_goal\_position** (*bool*) – Randomize goal position. Default = True.
- **p\_max\_step** (*int*) – Maximum step per episode. Default = 50.
- **p\_action\_type** (*int*) – Type of actions, which is either continuous action or discrete action. To be noted, discrete action is now limited to 2-d grid world. Default = C\_ACTION\_TYPE\_C.
- **p\_start\_position** (*dimension*) – To define the starting position, if p\_random\_start\_position is False, e.g. (3,2). Default = None.
- **p\_goal\_position** (*dimension*) – To define the goal position, if p\_random\_goal\_position is False, e.g. (5,5). Default = None.

**C\_NAME** = 'Grid World'

**C\_LATENCY** = `datetime.timedelta(seconds=1)`

**C\_INFINITY** = 3.4028235e+38

**C\_REWARD\_TYPE** = 0

**C\_ACTION\_TYPE\_CONT** = 0

**C\_ACTION\_TYPE\_DISC\_2D** = 1

**static setup\_spaces()**

Static template method to set up and return state and action space of environment.

**Returns**

- **state\_space** (*MSpace*) – State space object
- **action\_space** (*MSpace*) – Action space object

**\_setup\_spaces()**

**\_reset**(*p\_seed=None*) → None

To reset environment

**get\_all\_states()**

**\_simulate\_reaction**(*p\_state: State, p\_action: Action*) → *State*

Custom method for a simulated state transition. Implement this method if no external state transition function is used. See method `simulate_reaction()` for further details.

**\_compute\_reward**(*p\_state\_old: State, p\_state\_new: State*) → *Reward*

Custom reward method. See method `compute_reward()` for further details.

**\_compute\_success**(*p\_state: State*) → bool

Custom method for assessment for success. Implement this method if no external function is used. See method `compute_success()` for further details.



**\_compute\_broken**(*p\_state*: *State*) → bool

Custom method for assessment for breakdown. Implement this method if no external function is used. See method `compute_broken()` for further details.

**init\_plot**(*p\_figure*=None)

Initializes the plot functionalities of the class.

#### Parameters

- **p\_figure** (*Matplotlib.figure.Figure*, *optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p\_plot\_settings** (*PlotSettings*) – Optional plot settings. If None, the default view is plotted (see attribute `C_PLOT_DEFAULT_VIEW`).

**update\_plot**()

Updates the plot.

#### Parameters

**\*\*p\_kwargs** – Implementation-specific plot data and/or parameters.

## Multi-Cartpole

### Robot Manipulator

Ver. 1.1.9 (2023-08-21)

This module provides an environment of a robot manipulator based on Homogeneous Matrix

**class** `mlpro.rl.pool.envs.robotinhtm.RobotArm3D`

Bases: object

Auxiliary class for the implementation of `robotinhtm`. Generate the Kinematic of a pre-defined robot in Homogeneous Matrix.

**add\_link\_joint**(*jointAxis*=None, *lvector*=None, *thetaInit*=None, *adjustRot*=None, *adjustTheta*=None)

**get\_transformation\_matrix**(*theta*, *lvector*, *rotAxis*, *adjustRots*=None, *adjustThetas*=None)

**update\_joint\_coords**()

**get\_joint**()

**get\_num\_joint**()

**set\_theta**(*theta*)

**get\_homogeneous**()

**get\_homogeneous\_eef**()

**get\_orientation**()

**update\_theta**(*deltaTheta*)

```
class mlpro.rl.pool.envs.robotinhtm.RobotHTM(p_num_joints=4, p_reset_seed=True, p_seed=None,
                                             p_target_mode: str = 'random', p_visualize: bool = True,
                                             p_logging=True)
```

Bases: [Environment](#)

Custom environment for an arm robot represented as Homogeneous Matrix. The goal is to reach a certain point.

#### Parameters

- **p\_num\_joints** (*int*) – Number of joints. Default = 4.
- **p\_reset\_seed** (*bool*) – If True, random generator is reset. Default = True.
- **p\_seed** – Seeding value for the random generator. Default = None.
- **p\_target\_mode** (*str*) – Target mode. Possible values are “random” (default) or “fixed”.
- **p\_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = True.
- **p\_logging** – Log level (see constants of class Log). Default = Log.C\_LOG\_ALL.

**C\_NAME** = 'RobotHTM'

**C\_REWARD\_TYPE** = 0

**C\_LATENCY** = `datetime.timedelta(seconds=1)`

**C\_INFINITY** = 3.4028235e+38

**C\_CYCLE\_LIMIT** = 100

**static setup\_spaces()**

Static template method to set up and return state and action space of environment.

#### Returns

- **state\_space** (*MSpace*) – State space object
- **action\_space** (*MSpace*) – Action space object

**\_setup\_spaces()**

Implement this method to enrich the state and action space with specific dimensions.

**\_simulate\_reaction**(*p\_state*: [State](#), *p\_action*: [Action](#)) → [State](#)

Custom method for a simulated state transition. Implement this method if no external state transition function is used. See method `simulate_reaction()` for further details.

**\_compute\_success**(*p\_state*: [State](#) = None) → bool

Custom method for assessment for success. Implement this method if no external function is used. See method `compute_success()` for further details.

**\_compute\_broken**(*p\_state*: [State](#)) → bool

Custom method for assessment for breakdown. Implement this method if no external function is used. See method `compute_broken()` for further details.

**\_compute\_reward**(*p\_state\_old*: [State](#), *p\_state\_new*: [State](#)) → [Reward](#)

Custom reward method. See method `compute_reward()` for further details.

**set\_theta**(*theta*)

**\_reset**(*p\_seed=None*) → None

Custom method to reset the system to an initial/defined state. Use method `_set_status()` to set the state.

**Parameters**

**p\_seed** (*int*) – Seed parameter for an internal random generator

## Policies

### SARS-Buffers

#### Prioritized Buffer

Ver. 1.0.1 (2021-09-26)

This module provides the Prioritized Buffer based on the reference.

```
class mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.PrioritizedBufferElement(p_state: State,  
                                                                    p_action: Action,  
                                                                    p_reward: Reward,  
                                                                    p_state_new: State)
```

Bases: [\*SARSElement\*](#)

Element of a State-Action-Reward-Buffer.

```
class mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.PrioritizedBuffer(p_size=1, alpha: float = 0.3,  
                                                                    beta: float = 1)
```

Bases: [\*SARSBuffer\*](#)

Prioritized Sampling State-Action-Reward-Buffer in dictionary.

```
add_element(p_elem: PrioritizedBufferElement)
```

Add element to the buffer.

**Parameters**

**p\_elem** (*BufferElement*) – Element of Buffer

```
_gen_sample_ind(p_num: int) → list
```

Generate random indices from the buffer.

**Parameters**

**p\_num** (*int*) – Number of sample

**Returns**

List of indices

```
_extract_rows(p_list_idx: list)
```

Extract the element in the buffer based on a list of indices.

**Parameters**

**p\_list\_idx** (*list*) – List of indices

**Returns**

Samples in dictionary

```
get_latest()
```

Returns latest buffered element.

```
get_all()
```

Return all buffered elements.

**update\_priorities**(*p\_list\_idx: list, priorities: ndarray*)

Updates the priority tree. Needs to be called during each training step, utilising the element-wise calculated loss.

**class** mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.**SegmentTree**(*capacity: int, operation: Callable, init\_value: float*)

Bases: object

Reference: [https://github.com/openai/baselines/blob/master/baselines/common/segment\\_tree.py](https://github.com/openai/baselines/blob/master/baselines/common/segment_tree.py) .. attribute:: capacity

**type**  
int

**tree**

**Type**  
list

**operation**

**Type**  
function

**\_operate\_helper**(*start: int, end: int, node: int, node\_start: int, node\_end: int*) → float

Returns result of operation in segment.

**operate**(*start: int = 0, end: int = 0*) → float

Returns result of applying ‘self.operation’.

**class** mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.**SumSegmentTree**(*capacity: int*)

Bases: [SegmentTree](#)

Reference: [https://github.com/openai/baselines/blob/master/baselines/common/segment\\_tree.py](https://github.com/openai/baselines/blob/master/baselines/common/segment_tree.py)

**sum**(*start: int = 0, end: int = 0*) → float

Returns arr[start] + ... + arr[end].

**retrieve**(*upperbound: float*) → int

Find the highest index *i* about upper bound in the tree

**class** mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.**MinSegmentTree**(*capacity: int*)

Bases: [SegmentTree](#)

Reference: [https://github.com/openai/baselines/blob/master/baselines/common/segment\\_tree.py](https://github.com/openai/baselines/blob/master/baselines/common/segment_tree.py)

**min**(*start: int = 0, end: int = 0*) → float

Returns min(arr[start], ..., arr[end]).

## Random SARS Buffer

Ver. 1.0.0 (2021-09-25)

This module provides implementation of SARBuffer with random sampling.

**class** mlpro.rl.pool.sarsbuffer.RandomSARBuffer.**RandomSARBuffer**(*p\_size=1*)

Bases: BufferRnd

Random Sampling SARBuffer. This is just renaming class from BufferRnd

## 11.2.4 MLPro-GT - Game Theory

### Dynamic Games

#### Game Boards

#### Bulk Goods Plant

Ver. 1.0.7 (2023-06-27)

This module provides an environment of Bulk Good Laboratory Plant (BGLP) following GT interface. This module provides game board classed based on BGLP environment of the reinforcement learning pool.

```
class mlpro.gt.pool.boards.bglp.BGLP_GT(p_logging=True, t_step=0.5, t_set=10.0, demand=0.1,
                                         lr_margin=1.0, lr_demand=4.0, lr_power=0.001,
                                         margin_p=[0.2, 0.8, 4], prod_target=10000,
                                         prod_scenario='continuous', cycle_limit=100,
                                         p_visualize=False)
```

Bases: *BGLP*, *GameBoard*

Game theoretical pendant for the reinforcement learning environment class BGLP.

**C\_NAME** = 'BGLP\_GT'

### Multi-Cartpole

#### Native GT

#### Games

#### 2P Prisoners Dilemma

Ver. 1.0.2 (2024-01-27)

This module provides a 2-player game of Prisoners' Dilemma with random solver. In the near future, we are going to add more solvers and this howto is going to be updated accordingly.

The game consists of two competitors, where each competitor represents a prisoner. Both of them have a goal to minimize their prison sentences, where their length of sentences depend on their decision in front of the jury.

If a prisoner pleads guilty, while another prisoner pleads not guilty. The guilty prisoner gets 8 years of imprisonment, while the not guilty prisoner gets 1 year of imprisonment.

If both of them plead guilty, then each of them gets 2 years of imprisonment.

Meanwhile, if both of them plead not guilty, then each of them obtains 5 years of imprisonment.

To be noted, the decision making of the prisoners take place simultaneously, where: - Decision "0" means confess - Decision "1" means not confess

```
class mlpro.gt.pool.native.games.prisonersdilemma_2p.PayoffFunction_PD2P(p_func_type: int,
                                                                           p_dim_elems: list =
                                                                           None,
                                                                           p_num_coalitions:
                                                                           int = None,
                                                                           p_logging=True)
```

Bases: *GTFunction*

**\_setup\_mapping\_matrix()** → ndarray

A method to setup the mapping of the payoff matrix between strategies and payoffs. This is only applicable for C\_FUNC\_PAYOFF\_MATRIX. This method needs to be redefined based on the setup of the game.

**Returns**

Resulted mapping.

**Return type**

np.ndarray

**\_setup\_payoff\_matrix()**

A method to setup payoff matrices. This is only applicable for C\_FUNC\_PAYOFF\_MATRIX. This method needs to be redefined based on the setup of the game.

```
class mlpro.gt.pool.native.games.prisonersdilemma_2p.PrisonersDilemma2PGame(p_mode=0,
                                                                              p_ada=False,
                                                                              p_cycle_limit=1,
                                                                              p_visualize: bool
                                                                              = False,
                                                                              p_logging=True)
```

Bases: *GTGame*

**C\_NAME** = 'PrisonersDilemma2PGame'

**\_setup**(p\_mode, p\_ada: bool, p\_visualize: bool, p\_logging) → *Model*

Custom setup of GT Game. Payoff matrix has to be defined here as self.\_payoff.

**Parameters**

- **p\_mode** – Operation mode. See Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM
- **p\_ada** (bool) – Boolean switch for adaptivity.
- **p\_visualize** (bool) – Boolean switch for visualisation.
- **p\_logging** – Log level (see constants of class Log).

**Returns**

**player** – GTPlayer model (object of type GTPlayer, GTCoalition or GTCompetition).

**Return type**

*GTPlayer*

### 3P Prisoners Dilemma

Ver. 1.0.2 (2024-01-27)

This module provides a 2-player game of Prisoners' Dilemma with greedy and random solvers. In the near future, we are going to add more solvers and this howto is going to be updated accordingly.

The game consists of three competitors, where each competitor represents a prisoner. All of them have a goal to minimize their prison sentences, where their length of sentences depend on their decision in front of the jury.

If a prisoner pleads guilty, while another prisoner pleads not guilty. The guilty prisoner gets 10 years of imprisonment, while the not guilty prisoner gets 1 year of imprisonment.

If two of them plead guilty, then each of them gets 5 years of imprisonment, while the not guilty prisoner gets 1 year.

Meanwhile, if three of them plead not guilty, then each of them obtains 15 years of imprisonment.

And if three of them plead guilty, then each of them obtains 2 years of imprisonment.

To be noted, the decision making of the prisoners take place simultaneously, where: - Decision “0” means confess - Decision “1” means not confess

```
class mlpro.gt.pool.native.games.prisonersdilemma_3p.PayoffFunction_PD3P(p_func_type: int,
                                                                    p_dim_elems: list =
                                                                    None,
                                                                    p_num_coalitions:
                                                                    int = None,
                                                                    p_logging=True)
```

Bases: [GTFunction](#)

**\_setup\_mapping\_matrix()** → ndarray

A method to setup the mapping of the payoff matrix between strategies and payoffs. This is only applicable for C\_FUNC\_PAYOFF\_MATRIX. This method needs to be redefined based on the setup of the game.

**Returns**

Resulted mapping.

**Return type**

np.ndarray

**\_setup\_payoff\_matrix()**

A method to setup payoff matrices. This is only applicable for C\_FUNC\_PAYOFF\_MATRIX. This method needs to be redefined based on the setup of the game.

```
class mlpro.gt.pool.native.games.prisonersdilemma_3p.PrisonersDilemma3PGame(p_mode=0,
                                                                    p_ada=False,
                                                                    p_cycle_limit=1,
                                                                    p_visualize: bool
                                                                    = False,
                                                                    p_logging=True)
```

Bases: [GTGame](#)

**C\_NAME** = 'PrisonersDilemma3PGame'

**\_setup**(p\_mode, p\_ada: bool, p\_visualize: bool, p\_logging) → [Model](#)

Custom setup of GT Game. Payoff matrix has to be defined here as self.\_payoff.

**Parameters**

- **p\_mode** – Operation mode. See Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM
- **p\_ada** (bool) – Boolean switch for adaptivity.
- **p\_visualize** (bool) – Boolean switch for visualisation.
- **p\_logging** – Log level (see constants of class Log).

**Returns**

**player** – GTPlayer model (object of type GTPlayer, GTCoalition or GTCompetition).

**Return type**

[GTPlayer](#)

## Rock, Paper, Scissors

Ver. 1.0.1 (2024-01-12)

This module provides a duel of two coalitions of a game of Rock Paper Scissors with random solver. In the near future, we are going to add more solvers and this howto is going to be updated accordingly.

The game consists of two coalitions, where each coalition makes a decision based on the collaborative approach between the coalitions. Each coalition consists of 5 members, the most voted decision of the 5 members represents the final decision of the coalition.

To be noted, the decision making of the coalitions take place simultaneously, where: - Decision “0” means Rock - Decision “1” means Paper - Decision “2” means Scissors

```
class mlpro.gt.pool.native.games.rockpaperscissors.PayoffFunction_RSP(p_func_type: int,  
                                                                    p_dim_elems: list =  
                                                                    None, p_num_coalitions:  
                                                                    int = None,  
                                                                    p_logging=True)
```

Bases: *GTFunction*

**\_setup\_mapping\_matrix()** → ndarray

A method to setup the mapping of the payoff matrix between strategies and payoffs. This is only applicable for C\_FUNC\_PAYOFF\_MATRIX. This method needs to be redefined based on the setup of the game.

**Returns**

Resulted mapping.

**Return type**

np.ndarray

**\_setup\_payoff\_matrix()**

A method to setup payoff matrices. This is only applicable for C\_FUNC\_PAYOFF\_MATRIX. This method needs to be redefined based on the setup of the game.

```
class mlpro.gt.pool.native.games.rockpaperscissors.RockPaperScissors(p_mode=0, p_ada=False,  
                                                                    p_cycle_limit=1,  
                                                                    p_visualize: bool = False,  
                                                                    p_logging=True)
```

Bases: *GTGame*

**C\_NAME** = 'RockPaperScissors'

**\_setup**(p\_mode, p\_ada: bool, p\_visualize: bool, p\_logging) → *Model*

Custom setup of GT Game. Payoff matrix has to be defined here as self.\_payoff.

**Parameters**

- **p\_mode** – Operation mode. See Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM
- **p\_ada** (bool) – Boolean switch for adaptivity.
- **p\_visualize** (bool) – Boolean switch for visualisation.
- **p\_logging** – Log level (see constants of class Log).

**Returns**

**player** – GTPlayer model (object of type GTPlayer, GTCoalition or GTCompetition).



**Return type**  
*GTPlayer*

### 3P Routing Problems

Ver. 1.0.1 (2024-01-28)

This module provides a 3-player game of routing problems, where each player has to move simultaneously from the starting node to the target node. We also set up this game as a congestion game, which means that if multiple players select the same path, then the travelling time for both players are increased.

Node S represents the starting node, while node T denotes the target node. Here are the information of the routing network: Note -> [initial node to next node] : [x/y/z]

-> x = the travelling time, if only one player chooses this path -> y = the travelling time, if two players choose this path simultaneously -> z = the travelling time, if three players choose this path simultaneously

1. Node S to Node 1 : [4/6/10]
2. Node S to Node 2 : [3/4/5]
3. Node 1 to Node 2 : [1/2/5]
4. Node 1 to Node 3 : [3/5/6]
5. Node 2 to Node 3 : [4/5/6]
6. Node 2 to Node 4 : [3/6/9]
7. Node 3 to Node T : [2/4/6]
8. Node 4 to Node 3 : [1/2/7] 8. Node 4 to Node T : [2/8/10]

The main objective of each player is to reach the target points as fast as possible, while trying to avoid taking same actions with other players. This game represents a common scenario in industries, e.g. AGV routing plan, mobile robots, logistics, and many more.

**7 potential pathways can be selected by each player, such as:**

- 1) S -> 1 -> 2 -> 3 -> T
- 2) S -> 1 -> 3 -> T
- 3) S -> 1 -> 2 -> 4 -> 3 -> T
- 4) S -> 1 -> 2 -> 4 -> T
- 5) S -> 2 -> 4 -> T
- 6) S -> 2 -> 4 -> 3 -> T
- 7) S -> 2 -> 3 -> T

In this example, we are going to apply different solvers for each player, where Player 1 utilizes a min greedy policy, Player 2 utilizes a combination of a min greedy policy and a random policy, and Player 3 utilizes a random policy. In the near future, we are going to add more solvers and this game is going to be updated accordingly.

```
class mlpro.gt.pool.native.games.routingproblems_3p.TransferFunction_Routing3P(p_name: str,
                                                                              p_id: int =
                                                                              None,
                                                                              p_type: int =
                                                                              None,
                                                                              p_unit_in:
                                                                              str = None,
                                                                              p_unit_out:
                                                                              str = None,
                                                                              p_dt: float =
                                                                              0.01,
                                                                              p_logging=True,
                                                                              **p_args)
```

Bases: [TransferFunction](#)

**\_set\_function\_parameters**(p\_args) → bool

This method provides a functionality to set the parameters of the transfer function.

**Parameters**

**p\_args** (*dict*) – set of parameters of the transfer function.

**Returns**

true means no parameters are missing.

**Return type**

bool

**\_custom\_function**(p\_input, p\_range=None)

This function represents the template to create a custom function and must be redefined.

**Parameters**

- **p\_input** – input value.
- **p\_range** – range of the calculation. None means 0. Default: None.

**Returns**

output value.

**Return type**

float

```
class mlpro.gt.pool.native.games.routingproblems_3p.PayoffFunction_Routing3P(p_func_type:
                                                                              int,
                                                                              p_dim_elems:
                                                                              list = None,
                                                                              p_num_coalitions:
                                                                              int = None,
                                                                              p_logging=True)
```

Bases: [GTFunction](#)

**\_setup\_transfer\_functions**()

A method to setup transfer functions. This is only applicable for C\_FUNC\_TRANSFER\_FCTS. This method needs to be redefined based on the setup of the game.

```
class mlpro.gt.pool.native.games.routingproblems_3p.PayoffMatrix_Routing3P(p_function:
    GTFunction =
    None,
    p_player_ids: list
    = None,
    p_logging=True)
```

Bases: *GTPayoffMatrix*

**\_call\_mapping**(p\_input: str, p\_strategies: *GTStrategy*) → float

If the payoff matrix does not use the standardized matrix by MLPro-GT. This method can be used by redefining it.

**Parameters**

- **p\_input** (str) – inputs of the payoff matrix.
- **p\_strategies** (*GTStrategy*) – Selected strategies by all players/coalitions.

**Returns**

Payoff of the player/coalition.

**Return type**

float

**\_call\_best\_response**(p\_element\_id: str) → float

If the payoff matrix does not use the standardized matrix by MLPro-GT. This method can be used to get the best response value by redefining it.

**Parameters**

**p\_element\_id** (str) – Id of a specific player/coalition.

**Returns**

The highest possible payoff of a player/coalition.

**Return type**

float

**\_call\_zero\_sum**() → bool

If the payoff matrix does not use the standardized matrix by MLPro-GT. This method can be used to get the detect zero sum games by redefining it.

**Returns**

True means it is a zero sum game, otherwise no.

**Return type**

bool

```
class mlpro.gt.pool.native.games.routingproblems_3p.MinGreedyPolicy_Routing3P(p_strategy_space:
    MSpace,
    p_id=None,
    p_visualize:
    bool = False,
    p_logging=True,
    **p_param)
```

Bases: *MinGreedyPolicy*

**\_call\_compute\_strategy**(p\_payoff: *GTPayoffMatrix*) → *GTStrategy*

```
class mlpro.gt.pool.native.games.routingproblems_3p.Routing_3P(p_mode=0, p_ada=False,
                                                             p_cycle_limit=1, p_visualize:
                                                             bool = False, p_logging=True)
```

Bases: *GTGame*

**C\_NAME** = 'Routing\_3P'

**\_setup**(p\_mode, p\_ada: bool, p\_visualize: bool, p\_logging) → *Model*

Custom setup of GT Game. Payoff matrix has to be defined here as self.\_payoff.

#### Parameters

- **p\_mode** – Operation mode. See Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM
- **p\_ada** (bool) – Boolean switch for adaptivity.
- **p\_visualize** (bool) – Boolean switch for visualisation.
- **p\_logging** – Log level (see constants of class Log).

#### Returns

**player** – GTPlayer model (object of type GTPlayer, GTCoalition or GTCompetition).

#### Return type

*GTPlayer*

## 3P Supply and Demand

Ver. 1.0.1 (2024-01-12)

This module provides a 3-player game of supply and demand games, where each player represents a seller. Each seller has the capability to produce the same product with a quantity between 1-5 items everyday. The production cost of producing 1-5 items are constant, which is 5€. Therefore, the seller needs to sell an item with higher price, if they produce less amount.

The tables below show the sales price based on the quantity produced by each seller:

Seller 1 Seller 2 Seller 3

Price (€)	Quantity	Price (€)	Quantity	Price (€)	Quantity
15	1	10	1	8	1
12	2	8	2	7	2
9	3	6	3	6	3
6	4	4	4	4	4
5	5	4	5	4	5
2	5	4	5	2	5
4	5	2	5	4	5

The market demand of the products is 10 products/day. The buyer will always firstly buy the products with less prices. Therefore, each player needs an individual strategy in the competitive manner to select the quantity of the produced products in a day in order to maximize their profit.

In this example, we are going to apply different solvers for each seller, where Seller 1 and 2 utilize max greedy policy. This means that they always select the best possible outcomes without caring what the other sellers are doing. Meanwhile, Seller 3 utilized random solver. In the near future, we are going to add more solvers and this game is going to be updated accordingly.

```
class mlpro.gt.pool.native.games.supplydemand_3p.TransferFunction_SD3P(p_name: str, p_id: int
                                                                           = None, p_type: int =
                                                                           None, p_unit_in: str =
                                                                           None, p_unit_out: str =
                                                                           None, p_dt: float =
                                                                           0.01, p_logging=True,
                                                                           **p_args)
```

Bases: *TransferFunction*

**\_set\_function\_parameters**(*p\_args*) → bool

This method provides a functionality to set the parameters of the transfer function.

**Parameters**

**p\_args** (*dict*) – set of parameters of the transfer function.

**Returns**

true means no parameters are missing.

**Return type**

bool

**\_custom\_function**(*p\_input*, *p\_range=None*)

This function represents the template to create a custom function and must be redefined.

**Parameters**

- **p\_input** – input value.
- **p\_range** – range of the calculation. None means 0. Default: None.

**Returns**

output value.

**Return type**

float

```
class mlpro.gt.pool.native.games.supplydemand_3p.PayoffFunction_SD3P(p_func_type: int,
                                                                    p_dim_elems: list = None,
                                                                    p_num_coalisitions: int =
                                                                    None, p_logging=True)
```

Bases: [GTFunction](#)

**\_setup\_transfer\_functions**()

A method to setup transfer functions. This is only applicable for C\_FUNC\_TRANSFER\_FCTS. This method needs to be redefined based on the setup of the game.

```
class mlpro.gt.pool.native.games.supplydemand_3p.PayoffMatrix_SD3P(p_function: GTFunction =
                                                                    None, p_player_ids: list =
                                                                    None, p_logging=True)
```

Bases: [GTPayoffMatrix](#)

**\_call\_mapping**(*p\_input: str*, *p\_strategies: GTStrategy*) → float

If the payoff matrix does not use the standardized matrix by MLPro-GT. This method can be used by redefining it.

**Parameters**

- **p\_input** (*str*) – inputs of the payoff matrix.
- **p\_strategies** ([GTStrategy](#)) – Selected strategies by all players/coalitions.

**Returns**

Payoff of the player/coalition.

**Return type**

float

**\_call\_best\_response**(*p\_element\_id: str*) → float

If the payoff matrix does not use the standardized matrix by MLPro-GT. This method can be used to get the best response value by redefining it.

**Parameters**

**p\_element\_id** (*str*) – Id of a specific player/coalition.

**Returns**

The highest possible payoff of a player/coalition.

**Return type**

float

**\_call\_zero\_sum()** → bool

If the payoff matrix does not use the standardized matrix by MLPro-GT. This method can be used to get the detect zero sum games by redefining it.

**Returns**

True means it is a zero sum game, otherwise no.

**Return type**

bool

```
class mlpro.gt.pool.native.games.supplydemand_3p.MaxGreedyPolicy_SD3P_P1(p_strategy_space:
                                                                    MSpace, p_id=None,
                                                                    p_visualize: bool =
                                                                    False,
                                                                    p_logging=True,
                                                                    **p_param)
```

Bases: [MaxGreedyPolicy](#)

**\_call\_compute\_strategy**(*p\_payoff*: [GTPayoffMatrix](#)) → [GTStrategy](#)

```
class mlpro.gt.pool.native.games.supplydemand_3p.MaxGreedyPolicy_SD3P_P2(p_strategy_space:
                                                                    MSpace, p_id=None,
                                                                    p_visualize: bool =
                                                                    False,
                                                                    p_logging=True,
                                                                    **p_param)
```

Bases: [MaxGreedyPolicy](#)

**\_call\_compute\_strategy**(*p\_payoff*: [GTPayoffMatrix](#)) → [GTStrategy](#)

```
class mlpro.gt.pool.native.games.supplydemand_3p.SupplyDemand_3P(p_mode=0, p_ada=False,
                                                                    p_cycle_limit=1, p_visualize:
                                                                    bool = False, p_logging=True)
```

Bases: [GTGame](#)

**C\_NAME** = 'SupplyDemand\_3P'

**\_setup**(*p\_mode*, *p\_ada*: bool, *p\_visualize*: bool, *p\_logging*) → [Model](#)

Custom setup of GT Game. Payoff matrix has to be defined here as self.\_payoff.

**Parameters**

- **p\_mode** – Operation mode. See Mode.C\_VALID\_MODES for valid values. Default = Mode.C\_MODE\_SIM
- **p\_ada** (*bool*) – Boolean switch for adaptivity.
- **p\_visualize** (*bool*) – Boolean switch for visualisation.
- **p\_logging** – Log level (see constants of class Log).

**Returns****player** – GTPlayer model (object of type GTPlayer, GTCoalition or GTCompetition).**Return type***GTPlayer***Solvers****Greedy Policy**

Ver. 1.0.1 (2024-01-18)

This module provides solver with greedy GT strategy. There are two variants, such as minimum greedy and maximum greedy.

```
class mlpro.gt.pool.native.solvers.greedypolicy.MaxGreedyPolicy(p_strategy_space: MSpace,
                                                             p_id=None, p_visualize: bool =
                                                             False, p_logging=True,
                                                             **p_param)
```

Bases: *GTSolver*

A solver that generates actions for each dimension of the underlying strategy space based on the maximum greedy policy.

**C\_NAME** = 'MaxGreedyPolicy'**\_compute\_strategy**(p\_payoff: GTPayoffMatrix) → *GTStrategy*

A method to compute a strategy from the solver. This method needs to be redefined.

**Parameters****p\_payoff** (GTPayoffMatrix) – Payoff matrix of a specific player.**Returns**

The computed strategy.

**Return type***GTStrategy***\_call\_compute\_strategy**(p\_payoff: GTPayoffMatrix) → *GTStrategy*

```
class mlpro.gt.pool.native.solvers.greedypolicy.MinGreedyPolicy(p_strategy_space: MSpace,
                                                             p_id=None, p_visualize: bool =
                                                             False, p_logging=True,
                                                             **p_param)
```

Bases: *GTSolver*

A solver that generates actions for each dimension of the underlying strategy space based on the minimum greedy policy.

**C\_NAME** = 'MinGreedyPolicy'**\_compute\_strategy**(p\_payoff: GTPayoffMatrix) → *GTStrategy*

A method to compute a strategy from the solver. This method needs to be redefined.

**Parameters****p\_payoff** (GTPayoffMatrix) – Payoff matrix of a specific player.**Returns**

The computed strategy.

**Return type***GTStrategy***\_call\_compute\_strategy**(*p\_payoff*: *GTPayoffMatrix*) → *GTStrategy***Random Solver**

Ver. 1.0.1 (2024-01-12)

This module provides solver with random GT strategy.

```
class mlpro.gt.pool.native.solvers.randomsolver.RandomSolver(p_strategy_space: MSpace,  
                                                         p_id=None, p_visualize: bool =  
                                                         False, p_logging=True, **p_param)
```

Bases: *GTSolver*

A solver that generates random actions for each dimension of the underlying strategy space.

**C\_NAME** = 'RandomSolver'**\_compute\_strategy**(*p\_payoff*: *GTPayoffMatrix*) → *GTStrategy*

A method to compute a strategy from the solver. This method needs to be redefined.

**Parameters****p\_payoff** (*GTPayoffMatrix*) – Payoff matrix of a specific player.**Returns**

The computed strategy.

**Return type***GTStrategy***11.2.5 MLPro-OA - Online Adaptivity**

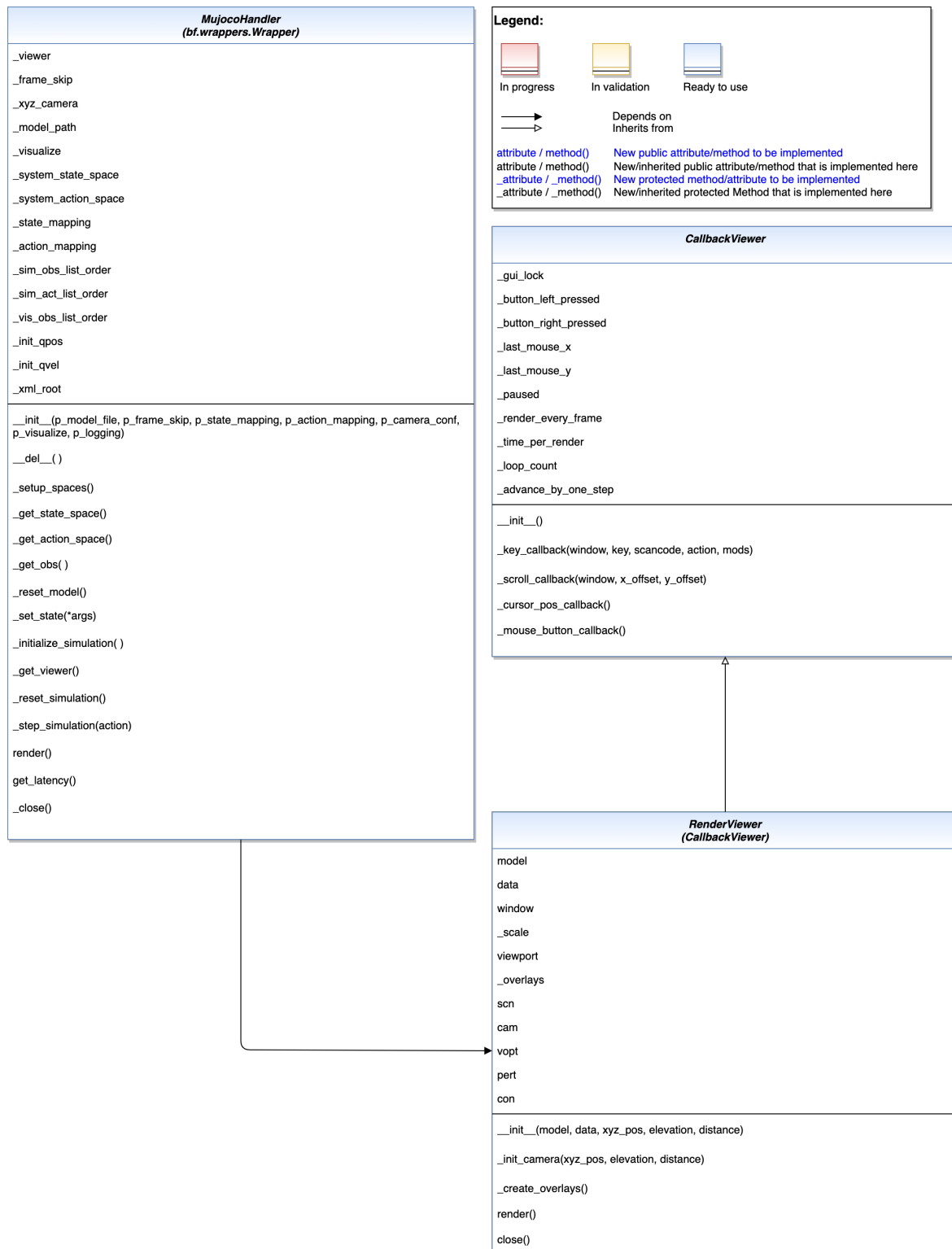
Coming soon...





### 11.3 3rd Party Support

### 11.3.1 MuJoCo



This module wraps bf.Systems with MuJoCo Simulation functionality.

**class** mlpro.wrappers.mujoco.CallbacksViewer

Bases: object

All callbacks function for the viewer

**\_key\_callback**(window, key, scancode, action, mods)

**\_scroll\_callback**(window, x\_offset, y\_offset)

**\_cursor\_pos\_callback**(window, xpos, ypos)

**\_mouse\_button\_callback**(window, button, act, mods)

**class** mlpro.wrappers.mujoco.BaseViewer(model, data, width, height)

Bases: object

**\_init\_camera**(xyz\_pos=None, elevation=None, distance=None)

**add\_overlay**(gridpos: int, text1: str, text2: str)

**\_set\_mujoco\_buffer**()

**\_make\_context\_current**()

**close**()

**class** mlpro.wrappers.mujoco.OffRenderViewer(model, data, xyz\_pos=None, elevation=None, distance=None)

Bases: [BaseViewer](#)

**\_set\_mujoco\_buffer**()

**\_make\_context\_current**()

**\_get\_opengl\_backend**(width: int, height: int)

**render**()

**close**()

**free**()

**class** mlpro.wrappers.mujoco.RenderViewer(model, data, xyz\_pos=None, elevation=None, distance=None)

Bases: [BaseViewer](#), [CallbacksViewer](#)

**\_create\_overlays**()

Should be user customizable

**\_set\_mujoco\_buffer**()

**\_make\_context\_current**()

**render**()

**close**()

**free**()

```
class mlpro.wrappers.mujoco.MujocoHandler(p_mujoco_file, p_frame_skip, p_state_mapping=None,
                                          p_action_mapping=None, p_camera_conf=(None, None,
                                          None), p_visualize=False, p_logging=True)
```

Bases: Wrapper

Module provides the functionality of MuJoCo.

#### Parameters

- **p\_mujoco\_file** (*str*) – String path points the MuJoCo file.
- **p\_frame\_skip** (*int*) – Frame skips for each simulation step.
- **p\_state\_mapping** (*list*) – State mapping for customized state. Defaults to None.
- **p\_action\_mapping** (*list*) – Action mapping for customized action. Defaults to None.
- **p\_camera\_conf** (*tuple*) – Camera configuration (xyz position, elevation, distance). Defaults to (None, None, None).
- **p\_visualize** (*bool*) – Visualize the MuJoCo Simulation. Defaults to False.
- **p\_logging** (*bool*) – Logging. Defaults to Log.C\_LOG\_ALL.

**C\_NAME** = 'MuJoCo'

**C\_TYPE** = 'Wrapper MuJoCo -> MLPro'

**C\_WRAPPED\_PACKAGE** = 'mujoco'

**C\_MINIMUM\_VERSION** = '2.3.1'

**setup\_spaces()**

Setup state and action spaces.

#### Returns

State Space and Action Space

#### Return type

*ESpace, ESpace*

**get\_init\_qpos\_space()**

Get Initial State Space

**get\_init\_qvel\_space()**

Get Initial State Space

**\_get\_state\_space()**

Generate the state space based on MJCF file.

#### Returns

State Space

#### Return type

*ESpace*

**\_get\_action\_space()**

Generate the action space based on MJCF file.

#### Returns

Action Space

#### Return type

*ESpace*

**\_get\_obs()**

Get the current observation of MuJoCo Simulation.

**Returns**

List of state values.

**Return type**

list

**\_reset\_model(reset\_state=None)**

Reset the model simulation. If `reset_state` is None, then the default state from MuJoCo will be taken for the initial value. Otherwise, a customized state can be defined in `_reset()` function.

**Parameters**

**reset\_state** (*list*) – qpos data and qvel data. Defaults to None.

**Returns**

List of state values

**Return type**

list

**\_set\_state(\*args)**

Set the current state of MuJoCo Simulation.

**\_initialize\_simulation()**

Initialize Simulation.

**\_setup\_camera(camera\_name)**

Setup MuJoCo Camera Object.

**Parameters**

**camera\_name** (*str*) – Camera name in MJCF file.

**Returns**

MuJoCo Camera Object.

**Return type**

`mujoco.MjvCamera`

**\_get\_camera\_data(cam)**

Get camera data from MuJoCo camera.

**Parameters**

**cam** (*mujoco.MjvCamera*) – MuJoCo Camera Object.

**Returns**

RGB Image or Depth Image.

**Return type**

ndarray

**\_get\_viewer()**

Get the MuJoCo viewer.

**Returns**

MuJoCo Viewer

**Return type**

*BaseViewer*

**`_reset_simulation(reset_state=None)`**

Reset the simulation. If `reset_state` is `None`, then the default state from MuJoCo will be taken for the initial value. Otherwise, a customized state can be defined in `_reset()` function.

**Parameters**

**`reset_state`** (*list*) – qpos data and qvel data. Defaults to `None`.

**Returns**

List of state values

**Return type**

list

**`_step_simulation(p_action: Action)`**

Pass the action to the simulation.

**Parameters**

**`p_action`** (*Action*) – Action

**`render()`**

Render the MuJoCo Viewer.

**`get_latency()`**

Get latency from MJCF file.

**Returns**

`None` or `Timestep`

**Return type**

float

**`_close()`**

Close MuJoCo Viewer

**Cross References**

Discover further integrated 3rd party packages in the MLPro Extension Hub.

## A3 - PROJECT MLPRO

### 12.1 Release Notes

**Release Notes on GitHub:**

- [Latest release](#)
- [Release history](#)

**Upcoming:**

- [Upcoming release](#)

### 12.2 Publications

#### 12.2.1 Papers

- Detlef Arend, Steve Yuwono, Mochammad Rizky Diprasetya, Andreas Schwung: “[MLPro 1.0 - Standardized reinforcement learning and game theory in Python](#)”, Machine Learning with Applications, 2022, ISSN 2666-8270, doi: 10.1016/j.mlwa.2022.100341
- Detlef Arend, Mochammad Rizky Diprasetya, Steve Yuwono, Andreas Schwung: “[MLPro—An integrative middleware framework for standardized machine learning tasks in Python](#)”, Software Impacts, 2022, doi: 10.1016/j.simpa.2022.100421

#### 12.2.2 Code Ocean Capsules

- Detlef Arend, Steve Yuwono, Mochammad Rizky Diprasetya, Andreas Schwung: “[MLPro - Reinforcement Learning - Demo of Advanced Training with Stagnation Detection](#)”, Code Ocean, 2022, doi: 10.24433/CO.9151382.v1
- Mochammad Rizky Diprasetya, Detlef Arend, Steve Yuwono, Andreas Schwung: “[MLPro - Reinforcement Learning - Demo of Training UR5 ROS Environment](#)”, Code Ocean, 2022, doi: 10.24433/CO.6279818.v1

### 12.2.3 GitHub Repository

- Detlef Arend, Steve Yuwono, Mochammad Rizky Diprasetya, Andreas Schwung: “MLPro - A Synoptic Framework for Standardized Machine Learning Tasks in Python”, Zenodo, 2021, doi: 10.5281/zenodo.6653485

## 12.3 Contribution

We look forward to your contributions to MLPro! Found a problem or have a good idea for an improvement? Just let us know directly on GitHub:

- [Problem Report](#)
- [Proposal for an improvement](#)

If you are interested in actively shaping MLPro, then simply contact us at: [mlpro@listen.fh-swf.de](mailto:mlpro@listen.fh-swf.de)

Our [templates](#) will help you create additional content in MLPro style. Of course, we will assist you in placing your contribution correctly.

We thank all [contributors](#) for their active support!

## 12.4 Disclaimer

If you require any more information or have any questions about our project’s disclaimer, please feel free to contact us by email at [mlpro@listen.fh-swf.de](mailto:mlpro@listen.fh-swf.de).

### 12.4.1 Disclaimers for MLPro Usage

All the information on this website is published in good faith and for general information purpose only. We do not make any warranties about the completeness, reliability and accuracy of this information. Any action you take upon the information you find on this project is strictly at your own risk. The South Westphalia University of Applied Sciences, Germany will not be liable for any losses and/or damages in connection with the use of MLPro.

From our website, you can visit other websites by following hyperlinks to such external sites. While we strive to provide only quality links to useful and ethical websites, we have no control over the content and nature of these sites. These links to other websites do not imply a recommendation for all the content found on these sites. Site owners and content may change without notice and may occur before we have the opportunity to remove a link which may have gone ‘bad’.

Please be also aware that when you leave our website, other sites may have different privacy policies and terms which are beyond our control. Please be sure to check the Privacy Policies of these sites as well as their “Terms of Service” before engaging in any business or uploading any information.

### 12.4.2 Consent

By using our website, you hereby consent to our disclaimer and agree to its terms.



## PYTHON MODULE INDEX

### m

- mlpro.bf.data, 278
- mlpro.bf.events, 293
- mlpro.bf.exceptions, 285
- mlpro.bf.math.basics, 308
- mlpro.bf.math.geometry, 315
- mlpro.bf.math.normalizers, 313
- mlpro.bf.ml.basics, 357
- mlpro.bf.ml.systems, 368
- mlpro.bf.mt, 296
- mlpro.bf.ops, 303
- mlpro.bf.physics.basics, 331
- mlpro.bf.physics.unitconverter, 335
- mlpro.bf.plot, 279
- mlpro.bf.streams.models, 318
- mlpro.bf.streams.samplers.min\_wise, 447
- mlpro.bf.streams.samplers.random, 448
- mlpro.bf.streams.samplers.reservoir\_sampling, 449
- mlpro.bf.streams.samplers.weighted\_random, 450
- mlpro.bf.streams.streams.clouds, 439
- mlpro.bf.streams.streams.csv\_file, 434
- mlpro.bf.streams.streams.doublespiral2d, 437
- mlpro.bf.streams.streams.point\_outliers, 438
- mlpro.bf.streams.streams.rnd10d, 436
- mlpro.bf.streams.tasks.deriver, 442
- mlpro.bf.streams.tasks.rearranger, 444
- mlpro.bf.streams.tasks.windows, 445
- mlpro.bf.systems.basics, 339
- mlpro.bf.systems.pool.doublependulum, 452
- mlpro.bf.ui.sciui.framework, 287
- mlpro.bf.ui.sciui.main, 285
- mlpro.bf.various, 271
- mlpro.gt.dynamicgames.basics, 398
- mlpro.gt.dynamicgames.potential, 400
- mlpro.gt.dynamicgames.stackelberg, 401
- mlpro.gt.native.basics, 404
- mlpro.gt.pool.boards.bglp, 501
- mlpro.gt.pool.native.games.prisonersdilemma\_2p, 501
- mlpro.gt.pool.native.games.prisonersdilemma\_3p, 502
- mlpro.gt.pool.native.games.rockpaperscissors, 504
- mlpro.gt.pool.native.games.routingproblems\_3p, 505
- mlpro.gt.pool.native.games.supplydemand\_3p, 508
- mlpro.gt.pool.native.solvers.greedypolicy, 511
- mlpro.gt.pool.native.solvers.randomsolver, 512
- mlpro.oa.streams.basics, 431
- mlpro.oa.streams.tasks.anomalydetectors, 431
- mlpro.oa.streams.tasks.boundarydetectors, 426
- mlpro.oa.streams.tasks.clusteranalyzers, 430
- mlpro.oa.streams.tasks.normalizers, 427
- mlpro.rl.models\_agents, 384
- mlpro.rl.models\_env, 375
- mlpro.rl.models\_env\_ada, 380
- mlpro.rl.models\_train, 391
- mlpro.rl.pool.actionplanner.mpc, 467
- mlpro.rl.pool.envs.bglp, 468
- mlpro.rl.pool.envs.doublependulum, 485
- mlpro.rl.pool.envs.gridworld, 495
- mlpro.rl.pool.envs.robotinhtml, 497
- mlpro.rl.pool.sarsbuffer.PrioritizedBuffer, 499
- mlpro.rl.pool.sarsbuffer.RandomSARSBBuffer, 500
- mlpro.sl.basics, 369
- mlpro.sl.fnn, 372
- mlpro.sl.pool.afct.fnn.pytorch.mlp, 461
- mlpro.sl.pool.afct.pytorch, 465
- mlpro.wrappers.mujoco, 514



## Symbols

<code>__cb_button_cancel()</code>	( <i>mlpro.bf.ui.sciui.framework.SciUISubplotSaveDLG method</i> ), 290	<code>_actions</code>	( <i>mlpro.bf.systems.basics.SystemShared attribute</i> ), 345
<code>__cb_button_folder()</code>	( <i>mlpro.bf.ui.sciui.framework.SciUISubplotSaveDLG method</i> ), 290	<code>_adapt()</code>	( <i>mlpro.bf.ml.basics.Model method</i> ), 360
<code>__cb_button_ok()</code>	( <i>mlpro.bf.ui.sciui.framework.SciUISubplotSaveDLG method</i> ), 290	<code>_adapt()</code>	( <i>mlpro.gt.dynamicgames.stackelberg.GTMultiPlayer_SG method</i> ), 402
<code>__cb_global_refresh()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 286	<code>_adapt()</code>	( <i>mlpro.gt.dynamicgames.stackelberg.GTPlayer_SG method</i> ), 401
<code>__cb_menu_file_properties()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 286	<code>_adapt()</code>	( <i>mlpro.oa.streams.basics.OATask method</i> ), 421, 432
<code>__cb_menu_help_about()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 286	<code>_adapt()</code>	( <i>mlpro.oa.streams.tasks.boundarydetectors.BoundaryDetector method</i> ), 426
<code>__cb_submenu_file_change_scenario()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 286	<code>_adapt()</code>	( <i>mlpro.oa.streams.tasks.normalizers.NormalizerZTransform method</i> ), 429
<code>__cb_toggle_fullscreen()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 286	<code>_adapt()</code>	( <i>mlpro.rl.models_agents.Agent method</i> ), 389
<code>__cb_window_resized()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 286	<code>_adapt()</code>	( <i>mlpro.rl.models_agents.MultiAgent method</i> ), 390
<code>__change_scenario()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 286	<code>_adapt()</code>	( <i>mlpro.rl.models_agents.Policy method</i> ), 385
<code>__init_cursor()</code>	( <i>mlpro.bf.ui.sciui.framework.SciUISubplot2D method</i> ), 291	<code>_adapt()</code>	( <i>mlpro.rl.models_env_ada.AFctReward method</i> ), 381
<code>__init_main_menu()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 285	<code>_adapt()</code>	( <i>mlpro.rl.models_env_ada.EnvModel method</i> ), 383
<code>__init_main_window()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 285	<code>_adapt()</code>	( <i>mlpro.sl.basics.SLAdaptiveFunction method</i> ), 370
<code>__init_shared_db()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 285	<code>_adapt_offline()</code>	( <i>mlpro.sl.basics.SLAdaptiveFunction method</i> ), 371
<code>__register_scenario_rec()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 286	<code>_adapt_offline()</code>	( <i>mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP method</i> ), 463
<code>__set_fullscreen()</code>	( <i>mlpro.bf.ui.sciui.main.SciUI method</i> ), 286	<code>_adapt_on_event()</code>	( <i>mlpro.bf.ml.basics.Model method</i> ), 360
<code>_action_dimensions</code>	( <i>mlpro.bf.systems.basics.SystemShared attribute</i> ), 345	<code>_adapt_on_event()</code>	( <i>mlpro.oa.streams.tasks.boundarydetectors.BoundaryDetector method</i> ), 426
<code>_action_to_mujoco()</code>	( <i>mlpro.bf.systems.basics.System method</i> ), 351	<code>_adapt_on_event()</code>	( <i>mlpro.oa.streams.tasks.normalizers.NormalizerMinMax method</i> ), 428
		<code>_adapt_online()</code>	( <i>mlpro.sl.basics.SLAdaptiveFunction method</i> ), 371
		<code>_adapt_online()</code>	( <i>mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP method</i> ), 463

`_adapt_policy_by_model()` (mlpro.rl.models\_agents.Agent method), 389  
`_adapt_reverse()` (mlpro.oa.streams.basics.OATask method), 421, 432  
`_adapt_reverse()` (mlpro.oa.streams.tasks.normalizers.NormalizerZTransform method), 429  
`_add_buffer()` (mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP method), 463  
`_add_init()` (mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP method), 463  
`_add_objective()` (mlpro.bf.ml.basics.Model method), 361  
`_add_payoff_matrix()` (mlpro.gt.native.basics.GTFunction method), 406  
`_add_transfer_function()` (mlpro.gt.native.basics.GTFunction method), 406  
`_afct_broken` (mlpro.rl.models\_env.EnvBase attribute), 378  
`_afct_reward` (mlpro.rl.models\_env.EnvBase attribute), 378  
`_afct_strans` (mlpro.rl.models\_env.EnvBase attribute), 378  
`_afct_success` (mlpro.rl.models\_env.EnvBase attribute), 378  
`_async_subtask()` (mlpro.rl.pool.actionplanner.mpc.MPC method), 468  
`_autorun()` (mlpro.bf.mt.Task method), 300  
`_calc_loss()` (mlpro.sl.fnn.FNN method), 373  
`_calc_loss()` (mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP method), 463  
`_call_best_response()` (mlpro.gt.native.basics.GTPayoffMatrix method), 408  
`_call_best_response()` (mlpro.gt.pool.native.games.routingproblems\_3p.PayoffMatrix\_Routing\_3P method), 507  
`_call_best_response()` (mlpro.gt.pool.native.games.supplydemand\_3p.PayoffMatrix\_SD3P method), 509  
`_call_compute_strategy()` (mlpro.gt.pool.native.games.routingproblems\_3p.MinGreedyPolicy\_Routing\_3P method), 507  
`_call_compute_strategy()` (mlpro.gt.pool.native.games.supplydemand\_3p.MaxGreedyPolicy\_SD3P\_P1 method), 510  
`_call_compute_strategy()` (mlpro.gt.pool.native.games.supplydemand\_3p.MaxGreedyPolicy\_SD3P\_P2 method), 510  
`_call_compute_strategy()` (mlpro.gt.pool.native.solvers.greedy\_policy.MaxGreedyPolicy method), 511  
`_call_compute_strategy()` (mlpro.gt.pool.native.solvers.greedy\_policy.MinGreedyPolicy method), 512  
`_call_mapping()` (mlpro.gt.native.basics.GTPayoffMatrix method), 407  
`_call_mapping()` (mlpro.gt.pool.native.games.routingproblems\_3p.PayoffMatrix\_Routing\_3P method), 507  
`_call_mapping()` (mlpro.gt.pool.native.games.supplydemand\_3p.PayoffMatrix\_SD3P method), 509  
`_call_zero_sum()` (mlpro.gt.native.basics.GTPayoffMatrix method), 408  
`_call_zero_sum()` (mlpro.gt.pool.native.games.routingproblems\_3p.PayoffMatrix\_Routing\_3P method), 507  
`_call_zero_sum()` (mlpro.gt.pool.native.games.supplydemand\_3p.PayoffMatrix\_SD3P method), 510  
`_close()` (mlpro.wrappers.mujoco.MujocoHandler method), 518  
`_close_episode()` (mlpro.rl.models\_train.RLTraining method), 397  
`_close_evaluation()` (mlpro.rl.models\_train.RLTraining method), 398  
`_close_results()` (mlpro.bf.ml.basics.Training method), 366  
`_close_trial()` (mlpro.gt.native.basics.GTTraining method), 419  
`_complete_state()` (mlpro.bf.ml.basics.Scenario method), 364  
`_complete_state()` (mlpro.bf.systems.basics.System method), 348  
`_complete_state()` (mlpro.bf.various.Persistent method), 374  
`_complete_state()` (mlpro.rl.models\_train.RLScenario method), 394  
`_complete_state()` (mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP method), 463  
`_compute_broken()` (mlpro.bf.systems.basics.FctBroken method), 352  
`_compute_broken()` (mlpro.bf.systems.basics.System method), 352  
`_compute_broken()` (mlpro.bf.systems.pool.doublependulum.DoublePendulumSystemRoom method), 455  
`_compute_broken()` (mlpro.rl.pool.envs.bglp.BGLP method), 455

<i>method</i> ), 482	<i>pro.wrappers.mujoco.RenderViewer method</i> ), 515
<code>_compute_broken()</code> ( <i>ml-pro.rl.pool.envs.gridworld.GridWorld method</i> ), 496	<code>_create_so()</code> ( <i>mlpro.bf.mt.Async method</i> ), 299
<code>_compute_broken()</code> ( <i>ml-pro.rl.pool.envs.robotinhtm.RobotHTM method</i> ), 498	<code>_cursor_pos_callback()</code> ( <i>ml-pro.wrappers.mujoco.CallbacksViewer method</i> ), 515
<code>_compute_reward()</code> ( <i>ml-pro.gt.dynamicgames.basics.GameBoard method</i> ), 399	<code>_custom_coalition_strategy()</code> ( <i>ml-pro.gt.native.basics.GTCoalition method</i> ), 413
<code>_compute_reward()</code> ( <i>mlpro.rl.models_env.FctReward method</i> ), 376	<code>_custom_function()</code> ( <i>ml-pro.bf.physics.basics.TransferFunction method</i> ), 333
<code>_compute_reward()</code> ( <i>ml-pro.rl.models_env_ada.AFctReward method</i> ), 381	<code>_custom_function()</code> ( <i>ml-pro.bf.streams.tasks.deriver.DerivativeFunction method</i> ), 443
<code>_compute_reward()</code> ( <i>mlpro.rl.pool.envs.bglp.BGLP method</i> ), 482	<code>_custom_function()</code> ( <i>ml-pro.gt.pool.native.games.routingproblems_3p.TransferFunction_F method</i> ), 506
<code>_compute_reward()</code> ( <i>ml-pro.rl.pool.envs.doublependulum.DoublePendulum method</i> ), 488	<code>_custom_function()</code> ( <i>ml-pro.gt.pool.native.games.supplydemand_3p.TransferFunction_SD method</i> ), 509
<code>_compute_reward()</code> ( <i>ml-pro.rl.pool.envs.gridworld.GridWorld method</i> ), 496	<code>_default_weight_bias_init()</code> ( <i>ml-pro.sl.pool.afct.pytorch.PyTorchHelperFunctions method</i> ), 467
<code>_compute_reward()</code> ( <i>ml-pro.rl.pool.envs.robotinhtm.RobotHTM method</i> ), 498	<code>_demand</code> ( <i>mlpro.rl.pool.envs.bglp.BGLP attribute</i> ), 478, 481
<code>_compute_strategy()</code> ( <i>ml-pro.gt.native.basics.GTSolver method</i> ), 409	<code>_derive_data()</code> ( <i>mlpro.bf.streams.tasks.deriver.Deriver method</i> ), 443
<code>_compute_strategy()</code> ( <i>ml-pro.gt.pool.native.solvers.greedypolicy.MaxGreedyPolicy method</i> ), 511	<code>_derivs()</code> ( <i>mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem method</i> ), 455
<code>_compute_strategy()</code> ( <i>ml-pro.gt.pool.native.solvers.greedypolicy.MinGreedyPolicy method</i> ), 511	<code>_export_action()</code> ( <i>mlpro.bf.systems.basics.System method</i> ), 351
<code>_compute_strategy()</code> ( <i>ml-pro.gt.pool.native.solvers.randomsolver.RandomSolver method</i> ), 512	<code>_extract_observation()</code> ( <i>ml-pro.rl.models_agents.Agent method</i> ), 389
<code>_compute_success()</code> ( <i>ml-pro.bf.systems.basics.FctSuccess method</i> ), 342	<code>_extract_rows()</code> ( <i>ml-pro.rl.pool.sarsbuffer.PrioritizedBuffer.PrioritizedBuffer method</i> ), 499
<code>_compute_success()</code> ( <i>mlpro.bf.systems.basics.System method</i> ), 351	<code>_fct_broken</code> ( <i>mlpro.bf.systems.basics.System attribute</i> ), 348
<code>_compute_success()</code> ( <i>ml-pro.bf.systems.pool.doublependulum.DoublePendulumSystem method</i> ), 455	<code>_fct_const()</code> ( <i>mlpro.bf.streams.streams.point_outliers.StreamMLProPOutliers method</i> ), 439
<code>_compute_success()</code> ( <i>mlpro.rl.pool.envs.bglp.BGLP method</i> ), 482	<code>_fct_cos()</code> ( <i>mlpro.bf.streams.streams.point_outliers.StreamMLProPOutliers method</i> ), 439
<code>_compute_success()</code> ( <i>ml-pro.rl.pool.envs.gridworld.GridWorld method</i> ), 496	<code>_fct_sin()</code> ( <i>mlpro.bf.streams.streams.point_outliers.StreamMLProPOutliers method</i> ), 439
<code>_compute_success()</code> ( <i>ml-pro.rl.pool.envs.robotinhtm.RobotHTM method</i> ), 498	<code>_fct_strans</code> ( <i>mlpro.bf.systems.basics.System attribute</i> ), 348
<code>_create_overlays()</code> ( <i>ml-</i>	<code>_fct_success</code> ( <i>mlpro.bf.systems.basics.System attribute</i> ), 348
	<code>_finalize_plot_view()</code> ( <i>ml-pro.bf.streams.models.StreamTask method</i> ), 326

`_force_fg()` (*mlpro.bf.plot.Plottable* method), 282  
`_function_approximation()` (*mlpro.bf.physics.basics.TransferFunction* method), 334  
`_gen_current_path()` (*mlpro.bf.ml.basics.Training* method), 366  
`_gen_root_path()` (*mlpro.bf.ml.basics.Training* method), 366  
`_gen_sample_ind()` (*mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.PrioritizedBuffer* method), 499  
`_get_accuracy()` (*mlpro.bf.ml.basics.Model* method), 361  
`_get_action_space()` (*mlpro.wrappers.mujoco.MujocoHandler* method), 516  
`_get_camera_data()` (*mlpro.wrappers.mujoco.MujocoHandler* method), 517  
`_get_custom_run_method()` (*mlpro.bf.mt.Task* method), 300  
`_get_custom_run_method()` (*mlpro.bf.streams.models.StreamTask* method), 325  
`_get_evaluation()` (*mlpro.gt.native.basics.GTGame* method), 416  
`_get_next()` (*mlpro.bf.streams.models.Stream* method), 323  
`_get_next()` (*mlpro.bf.streams.streams.clouds.StreamMLProClouds* method), 440  
`_get_next()` (*mlpro.bf.streams.streams.csv\_file.StreamMLProCSV* method), 435  
`_get_next()` (*mlpro.bf.streams.streams.point\_outliers.StreamMLProPOutliers* method), 439  
`_get_next()` (*mlpro.bf.streams.streams.rnd10d.StreamMLProRnd10D* method), 437  
`_get_next_action()` (*mlpro.bf.systems.basics.DemoScenario* method), 355  
`_get_obs()` (*mlpro.wrappers.mujoco.MujocoHandler* method), 517  
`_get_opengl_backend()` (*mlpro.wrappers.mujoco.OffRenderViewer* method), 515  
`_get_path()` (*mlpro.bf.various.Persistent* method), 274  
`_get_plot_host_task()` (*mlpro.bf.mt.Workflow* method), 302  
`_get_sensor_value()` (*mlpro.bf.systems.basics.Controller* method), 344  
`_get_state_space()` (*mlpro.wrappers.mujoco.MujocoHandler* method), 516  
`_get_stream()` (*mlpro.bf.streams.models.StreamProvider* method), 324  
`_get_stream_list()` (*mlpro.bf.streams.models.StreamProvider* method), 323  
`_get_viewer()` (*mlpro.wrappers.mujoco.MujocoHandler* method), 517  
`_hyperparam_handler()` (*mlpro.bf.ml.basics.Model* method), 359  
`_import_state()` (*mlpro.bf.systems.basics.System* method), 351  
`_init_camera()` (*mlpro.wrappers.mujoco.BaseViewer* method), 515  
`_init_dataset()` (*mlpro.bf.streams.streams.clouds.StreamMLProClouds* method), 440  
`_init_dataset()` (*mlpro.bf.streams.streams.csv\_file.StreamMLProCSV* method), 435  
`_init_dataset()` (*mlpro.bf.streams.streams.doublespiral2d.DoubleSpiral2D* method), 437  
`_init_dataset()` (*mlpro.bf.streams.streams.point\_outliers.StreamMLProPOutliers* method), 438  
`_init_dataset()` (*mlpro.bf.streams.streams.rnd10d.StreamMLProRnd10D* method), 436  
`_init_episode()` (*mlpro.rl.models\_train.RLTraining* method), 397  
`_init_evaluation()` (*mlpro.rl.models\_train.RLTraining* method), 397  
`_init_figure()` (*mlpro.bf.plot.Plottable* method), 281  
`_init_figure()` (*mlpro.bf.streams.models.StreamScenario* method), 329  
`_init_hyperparam()` (*mlpro.bf.ml.basics.Model* method), 359  
`_init_hyperparam()` (*mlpro.gt.native.basics.GTPlayer* method), 410  
`_init_hyperparam()` (*mlpro.gt.native.basics.GTSolver* method), 408  
`_init_hyperparam()` (*mlpro.rl.models\_agents.Agent* method), 388  
`_init_hyperparam()` (*mlpro.rl.models\_env\_ada.EnvModel* method), 382  
`_init_hyperparam()` (*mlpro.sl.fnn.MLP* method), 373  
`_init_hyperparam()` (*mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP* method), 462  
`_init_plot_2d()` (*mlpro.bf.plot.Plottable* method), 282  
`_init_plot_2d()` (*mlpro.bf.streams.models.StreamTask* method), 325



<code>_init_plot_2d()</code> (mlpro.bf.streams.models.StreamWorkflow method), 327	<code>_iterator</code> (mlpro.oa.streams.basics.OAScenario attribute), 423
<code>_init_plot_2d()</code> (mlpro.bf.streams.tasks.windows.Window method), 446	<code>_key_callback()</code> (mlpro.wrappers.mujoco.CallbacksViewer method), 515
<code>_init_plot_2d()</code> (mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem method), 456	<code>_last_action</code> (mlpro.bf.systems.basics.System attribute), 347
<code>_init_plot_2d()</code> (mlpro.rl.pool.envs.doublependulum.DoublePendulumSystem method), 489	<code>_last_action</code> (mlpro.rl.models_env.EnvBase attribute), 377
<code>_init_plot_3d()</code> (mlpro.bf.plot.Plottable method), 282	<code>_last_reward</code> (mlpro.rl.models_env.EnvBase attribute), 378
<code>_init_plot_3d()</code> (mlpro.bf.streams.models.StreamTask method), 326	<code>_latency</code> (mlpro.bf.systems.basics.System attribute), 347
<code>_init_plot_3d()</code> (mlpro.bf.streams.models.StreamWorkflow method), 327	<code>_latency</code> (mlpro.rl.models_env.EnvBase attribute), 377
<code>_init_plot_3d()</code> (mlpro.bf.streams.tasks.windows.Window method), 446	<code>_linear()</code> (mlpro.bf.physics.basics.TransferFunction method), 333
<code>_init_plot_nd()</code> (mlpro.bf.plot.Plottable method), 282	<code>_log_results()</code> (mlpro.bf.ml.basics.TrainingResults method), 364
<code>_init_plot_nd()</code> (mlpro.bf.streams.models.StreamTask method), 326	<code>_log_results()</code> (mlpro.rl.models_train.RLTrainingResults method), 396
<code>_init_plot_nd()</code> (mlpro.bf.streams.models.StreamWorkflow method), 327	<code>_make_context_current()</code> (mlpro.wrappers.mujoco.BaseViewer method), 515
<code>_init_plot_nd()</code> (mlpro.bf.streams.tasks.windows.Window method), 446	<code>_make_context_current()</code> (mlpro.wrappers.mujoco.OffRenderViewer method), 515
<code>_init_plot_nd()</code> (mlpro.bf.streams.models.StreamWorkflow method), 327	<code>_make_context_current()</code> (mlpro.wrappers.mujoco.RenderViewer method), 515
<code>_init_plot_nd()</code> (mlpro.bf.streams.tasks.windows.Window method), 446	<code>_map()</code> (mlpro.bf.math.basics.Function method), 312
<code>_init_plot_nd()</code> (mlpro.oa.streams.tasks.boundarydetectors.BoundaryDetector method), 427	<code>_map()</code> (mlpro.bf.systems.basics.SystemShared method), 346
<code>_init_results()</code> (mlpro.bf.ml.basics.TrainingResults method), 366	<code>_map()</code> (mlpro.sl.fnn.FNN method), 373
<code>_init_results()</code> (mlpro.gt.native.basics.GTTraining method), 418	<code>_map()</code> (mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP method), 463
<code>_init_results()</code> (mlpro.rl.models_train.RLTraining method), 397	<code>_map_values()</code> (mlpro.bf.systems.basics.SystemShared method), 345
<code>_init_timer()</code> (mlpro.bf.ops.ScenarioBase method), 305	<code>_mappings</code> (mlpro.bf.systems.basics.SystemShared attribute), 345
<code>_init_trial()</code> (mlpro.gt.native.basics.GTTraining method), 418	<code>_maximize()</code> (mlpro.bf.ml.basics.HyperParamTuner method), 365
<code>_initialize_simulation()</code> (mlpro.wrappers.mujoco.MujocoHandler method), 517	<code>_mouse_button_callback()</code> (mlpro.wrappers.mujoco.CallbacksViewer method), 515
<code>_input_preproc()</code> (mlpro.sl.pool.afct.pytorch.PyTorchHelperFunctions method), 466	<code>_normalize()</code> (mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem method), 456
<code>_instances</code> (mlpro.bf.streams.models.StreamShared attribute), 319	<code>_normalize()</code> (mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem method), 458
<code>_is_bestresponse()</code> (mlpro.gt.native.basics.GTGame method), 419	<code>_normalize()</code> (mlpro.rl.pool.envs.doublependulum.DoublePendulumS4 method), 491
	<code>_obs_to_mujoco()</code> (mlpro.bf.systems.pool.doublependulum.DoublePendulumSystemS4 method), 458
	<code>_obs_to_mujoco()</code> (mlpro.rl.pool.envs.doublependulum.DoublePendulumS4 method), 491

`pro.rl.pool.envs.doublependulum.DoublePendulumS4` 423  
`method)`, 491  
`_omit_instance()` (`mlpro.bf.streams.models.Sampler` `method)`, 321  
`_omit_instance()` (`ml-` `method)`, 363  
`pro.bf.streams.samplers.min_wise.SamplerMinWise` `method)`, 448  
`_omit_instance()` (`ml-` `method)`, 348  
`pro.bf.streams.samplers.random.SamplerRND` `method)`, 449  
`_omit_instance()` (`ml-` `method)`, 394  
`pro.bf.streams.samplers.reservoir_sampling.SamplerReservoir` `method)`, 450  
`_omit_instance()` (`ml-` `method)`, 464  
`pro.bf.streams.samplers.weighted_random.SamplerWeightedRND` `method)`, 451  
`_operate_helper()` (`ml-` `method)`, 317  
`pro.rl.pool.sarsbuffer.PrioritizedBuffer.SegmentTree` `method)`, 500  
`_optimize()` (`mlpro.sl.fnn.FNN` `method)`, 373  
`_optimize()` (`mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP` `method)`, 463  
`_output_postproc()` (`ml-` `method)`, 467  
`pro.sl.pool.afct.pytorch.PyTorchHelperFunctions` `method)`, 467  
`_output_preproc()` (`ml-` `method)`, 467  
`pro.sl.pool.afct.pytorch.PyTorchHelperFunctions` `method)`, 467  
`_plan_action()` (`mlpro.rl.models_agents.ActionPlanner` `method)`, 386  
`_plan_action()` (`mlpro.rl.pool.actionplanner.mpc.MPC` `method)`, 467  
`_plot_settings` (`mlpro.oa.streams.basics.OAScenario` `attribute)`, 423  
`_plot_settings` (`mlpro.oa.streams.basics.OATask` `attribute)`, 422  
`_plot_settings` (`mlpro.oa.streams.basics.OAWorkflow` `attribute)`, 423  
`_prepare_rearrangement()` (`ml-` `method)`, 444  
`pro.bf.streams.tasks.rearranger.Rearranger` `method)`, 444  
`_prev_state` (`mlpro.bf.systems.basics.System` `attribute)`, 347  
`_process_action()` (`mlpro.bf.systems.basics.System` `method)`, 350  
`_process_action()` (`mlpro.rl.models_env.EnvBase` `method)`, 378  
`_process_action()` (`ml-` `method)`, 383  
`pro.rl.models_env_ada.EnvModel` `method)`, 383  
`_raise_event()` (`mlpro.bf.events.EventManager` `method)`, 294  
`_range` (`mlpro.oa.streams.basics.OATask` `attribute)`, 422  
`_range` (`mlpro.oa.streams.basics.OAWorkflow` `attribute)`, 422  
`_rearrange()` (`mlpro.bf.streams.tasks.rearranger.Rearranger` `method)`, 444  
`_reduce_state()` (`mlpro.bf.ml.basics.Scenario` `method)`, 363  
`_reduce_state()` (`mlpro.bf.systems.basics.System` `method)`, 348  
`_reduce_state()` (`mlpro.bf.various.Persistent` `method)`, 274  
`_reduce_state()` (`mlpro.rl.models_train.RLScenario` `method)`, 394  
`_reduce_state()` (`ml-` `method)`, 464  
`pro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP` `method)`, 464  
`_refresh_plot()` (`mlpro.bf.plot.Plottable` `method)`, 282  
`_remove_plot_2d()` (`mlpro.bf.math.geometry.Point` `method)`, 317  
`_remove_plot_2d()` (`mlpro.bf.plot.Plottable` `method)`, 283  
`_remove_plot_3d()` (`mlpro.bf.math.geometry.Point` `method)`, 317  
`_remove_plot_3d()` (`mlpro.bf.plot.Plottable` `method)`, 283  
`_remove_plot_nd()` (`mlpro.bf.math.geometry.Point` `method)`, 317  
`_remove_plot_nd()` (`mlpro.bf.plot.Plottable` `method)`, 283  
`_renormalize()` (`mlpro.oa.streams.basics.OATask` `method)`, 422, 432  
`_reset()` (`mlpro.bf.ops.ScenarioBase` `method)`, 305  
`_reset()` (`mlpro.bf.streams.models.Stream` `method)`, 322  
`_reset()` (`mlpro.bf.streams.models.StreamScenario` `method)`, 329  
`_reset()` (`mlpro.bf.streams.streams.csv_file.StreamMLProCSV` `method)`, 435  
`_reset()` (`mlpro.bf.systems.basics.Controller` `method)`, 343  
`_reset()` (`mlpro.bf.systems.basics.DemoScenario` `method)`, 355  
`_reset()` (`mlpro.bf.systems.basics.MultiSystem` `method)`, 353  
`_reset()` (`mlpro.bf.systems.basics.System` `method)`, 349  
`_reset()` (`mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem` `method)`, 455  
`_reset()` (`mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem` `method)`, 460  
`_reset()` (`mlpro.rl.models_env_ada.EnvModel` `method)`, 383  
`_reset()` (`mlpro.rl.models_train.RLScenario` `method)`, 395  
`_reset()` (`mlpro.rl.pool.envs.bglp.BGLP` `method)`, 482  
`_reset()` (`mlpro.rl.pool.envs.gridworld.GridWorld` `method)`, 496  
`_reset()` (`mlpro.rl.pool.envs.robotinhtm.RobotHTM`



method), 498

`_reset_model()` (`mlpro.wrappers.mujoco.MujocoHandler` `set_adapted()` (`mlpro.bf.ml.basics.Model` method), method), 517

`_reset_simulation()` (`mlpro.wrappers.mujoco.MujocoHandler` method), 517

`_run()` (`mlpro.bf.ml.basics.Training` method), 367

`_run()` (`mlpro.bf.mt.Task` method), 301

`_run()` (`mlpro.bf.streams.models.StreamTask` method), 325

`_run()` (`mlpro.bf.streams.tasks.deriver.Deriver` method), 443

`_run()` (`mlpro.bf.streams.tasks.rearranger.Rearranger` method), 444

`_run()` (`mlpro.bf.streams.tasks.windows.Window` method), 445

`_run()` (`mlpro.bf.systems.basics.System` method), 350

`_run()` (`mlpro.oa.streams.tasks.boundarydetectors.BoundaryDetector` method), 508

`_run()` (`mlpro.oa.streams.tasks.boundarydetectors.BoundaryDetector` method), 426

`_run()` (`mlpro.oa.streams.tasks.normalizers.NormalizerMinMax` method), 428

`_run()` (`mlpro.oa.streams.tasks.normalizers.NormalizerZTransform` method), 429

`_run_async()` (`mlpro.bf.mt.Task` method), 301

`_run_cycle()` (`mlpro.bf.ml.basics.Training` method), 366

`_run_cycle()` (`mlpro.bf.ops.ScenarioBase` method), 306

`_run_cycle()` (`mlpro.bf.streams.models.StreamScenario` method), 329

`_run_cycle()` (`mlpro.bf.systems.basics.DemoScenario` method), 355

`_run_cycle()` (`mlpro.gt.native.basics.GTGame` method), 416

`_run_cycle()` (`mlpro.gt.native.basics.GTTraining` method), 419

`_run_cycle()` (`mlpro.rl.models_train.RLScenario` method), 395

`_run_cycle()` (`mlpro.rl.models_train.RLTraining` method), 398

`_run_wrapper()` (`mlpro.bf.streams.models.StreamTask` method), 325

`_save_line()` (`mlpro.bf.ml.basics.HyperParamTuner` method), 365

`_save_line()` (`mlpro.bf.ml.basics.TrainingResults` method), 364

`_scalar_conversion()` (`mlpro.bf.physics.unitconverter.UnitConverter` method), 337

`_scroll_callback()` (`mlpro.wrappers.mujoco.CallbacksViewer` method), 515

`_set_actuator_value()` (`mlpro.bf.systems.basics.Controller` method), 344

`_set_function_parameters()` (`mlpro.bf.physics.basics.TransferFunction` method), 333

`_set_function_parameters()` (`mlpro.bf.physics.unitconverter.UnitConverter` method), 337

`_set_function_parameters()` (`mlpro.bf.streams.tasks.deriver.DerivativeFunction` method), 443

`_set_function_parameters()` (`mlpro.gt.pool.native.games.routingproblems_3p.TransferFunction_K` method), 506

`_set_function_parameters()` (`mlpro.gt.pool.native.games.supplydemand_3p.TransferFunction_SD` method), 506

`_set_mode()` (`mlpro.bf.ops.ScenarioBase` method), 305

`_set_mode()` (`mlpro.bf.streams.models.StreamScenario` method), 328

`_set_mode()` (`mlpro.rl.models_train.RLScenario` method), 395

`_set_mujoco_buffer()` (`mlpro.wrappers.mujoco.BaseViewer` method), 515

`_set_mujoco_buffer()` (`mlpro.wrappers.mujoco.OffRenderViewer` method), 515

`_set_mujoco_buffer()` (`mlpro.wrappers.mujoco.RenderViewer` method), 515

`_set_parameters()` (`mlpro.bf.math.normalizers.Normalizer` method), 313

`_set_state()` (`mlpro.bf.systems.basics.System` method), 350

`_set_state()` (`mlpro.wrappers.mujoco.MujocoHandler` method), 517

`_set_type()` (`mlpro.bf.physics.basics.TransferFunction` method), 333

`_setup()` (`mlpro.bf.ml.basics.Scenario` method), 363

`_setup()` (`mlpro.bf.streams.models.StreamScenario` method), 328

`_setup()` (`mlpro.gt.native.basics.GTGame` method), 416

`_setup()` (`mlpro.gt.pool.native.games.prisonersdilemma_2p.PrisonersDilemma` method), 502

`_setup()` (`mlpro.gt.pool.native.games.prisonersdilemma_3p.PrisonersDilemma` method), 503

`_setup()` (`mlpro.gt.pool.native.games.rockpaperscissors.RockPaperScissors` method), 504

`_setup()` (`mlpro.gt.pool.native.games.routingproblems_3p.Routing_3P` method), 508

`_setup()` (`mlpro.gt.pool.native.games.supplydemand_3p.SupplyDemand_3P` method), 508

method), 510

`_setup()` (*mlpro.oa.streams.basics.OAScenario* method), 423, 433

`_setup()` (*mlpro.rl.models\_agents.ActionPlanner* method), 386

`_setup()` (*mlpro.rl.models\_agents.RLScenarioMBSInt* method), 387

`_setup()` (*mlpro.rl.models\_train.RLScenario* method), 394

`_setup_camera()` (*ml-pro.wrappers.mujoco.MujocoHandler* method), 517

`_setup_feature_space()` (*ml-pro.bf.streams.models.Stream* method), 322

`_setup_feature_space()` (*ml-pro.bf.streams.streams.clouds.StreamMLProClouds* method), 440

`_setup_feature_space()` (*ml-pro.bf.streams.streams.csv\_file.StreamMLProCSV* method), 435

`_setup_feature_space()` (*ml-pro.bf.streams.streams.doublespiral2d.DoubleSpiral2D* method), 437

`_setup_feature_space()` (*ml-pro.bf.streams.streams.point\_outliers.StreamMLProPOutliers* method), 438

`_setup_feature_space()` (*ml-pro.bf.streams.streams.rnd10d.StreamMLProRnd10D* method), 436

`_setup_label_space()` (*ml-pro.bf.streams.models.Stream* method), 322

`_setup_label_space()` (*ml-pro.bf.streams.streams.csv\_file.StreamMLProCSV* method), 435

`_setup_label_space()` (*ml-pro.bf.streams.streams.rnd10d.StreamMLProRnd10D* method), 436

`_setup_logging_set()` (*ml-pro.sl.basics.SLAdaptiveFunction* method), 370

`_setup_mapping_matrix()` (*ml-pro.gt.native.basics.GTFunction* method), 405

`_setup_mapping_matrix()` (*ml-pro.gt.pool.native.games.prisonersdilemma\_2p.PayoffFunction\_2P* method), 502

`_setup_mapping_matrix()` (*ml-pro.gt.pool.native.games.prisonersdilemma\_3p.PayoffFunction\_3P* method), 503

`_setup_mapping_matrix()` (*ml-pro.gt.pool.native.games.rockpaperscissors.PayoffFunction\_RSP* method), 504

`_setup_model()` (*mlpro.sl.basics.SLAdaptiveFunction* method), 370

`_setup_model()` (*mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP* method), 462

`_setup_payoff_matrix()` (*ml-pro.gt.native.basics.GTFunction* method), 406

`_setup_payoff_matrix()` (*ml-pro.gt.pool.native.games.prisonersdilemma\_2p.PayoffFunction\_2P* method), 502

`_setup_payoff_matrix()` (*ml-pro.gt.pool.native.games.prisonersdilemma\_3p.PayoffFunction\_3P* method), 503

`_setup_payoff_matrix()` (*ml-pro.gt.pool.native.games.rockpaperscissors.PayoffFunction\_RSP* method), 504

`_setup_solver()` (*mlpro.gt.native.basics.GTSolver* method), 409

`_setup_spaces()` (*ml-pro.rl.models\_env\_ada.AFctReward* method), 381

`_setup_spaces()` (*ml-pro.rl.pool.envs.gridworld.GridWorld* method), 496

`_setup_spaces()` (*ml-pro.rl.pool.envs.robotinhtm.RobotHTM* method), 498

`_setup_transfer_functions()` (*ml-pro.gt.native.basics.GTFunction* method), 406

`_setup_transfer_functions()` (*ml-pro.gt.pool.native.games.routingproblems\_3p.PayoffFunction\_Ro* method), 506

`_setup_transfer_functions()` (*ml-pro.gt.pool.native.games.supplydemand\_3p.PayoffFunction\_SD3P* method), 509

`_simulate_reaction()` (*ml-pro.bf.systems.basics.FctSTrans* method), 341

`_simulate_reaction()` (*ml-pro.bf.systems.basics.System* method), 351

`_simulate_reaction()` (*ml-pro.bf.systems.pool.doublependulum.DoublePendulumSystemRo* method), 455

`_simulate_reaction()` (*ml-pro.bf.systems.pool.doublependulum.DoublePendulumSystemS7* method), 460

`_simulate_reaction()` (*ml-pro.rl.pool.envs.bglp.BGLP* method), 482

`_simulate_reaction()` (*ml-pro.rl.pool.envs.gridworld.GridWorld* method), 496

`_simulate_reaction()` (*ml-pro.rl.pool.envs.robotinhtm.RobotHTM* method), 498

`_so` (*mlpro.oa.streams.basics.OATask* attribute), 422

- `_so` (*mlpro.oa.streams.basics.OAWorkflow* attribute), 423
  - `_spaces` (*mlpro.bf.systems.basics.SystemShared* attribute), 345
  - `_start_async` (*mlpro.bf.mt.Async* method), 299
  - `_state` (*mlpro.bf.systems.basics.System* attribute), 347
  - `_state` (*mlpro.rl.models\_env.EnvBase* attribute), 377
  - `_state_from_mujoco` (*mlpro.bf.systems.basics.System* method), 351
  - `_states` (*mlpro.bf.systems.basics.SystemShared* attribute), 345
  - `_step_simulation` (*mlpro.wrappers.mujoco.MujocoHandler* method), 518
  - `_stream` (*mlpro.oa.streams.basics.OAScenario* attribute), 423
  - `_temperature` (*mlpro.bf.physics.unitconverter.UnitConverter* method), 338
  - `_update_evaluation` (*mlpro.rl.models\_train.RLTraining* method), 397
  - `_update_hyperparameters` (*mlpro.bf.ml.basics.Model* method), 359
  - `_update_plot_2d` (*mlpro.bf.math.geometry.Point* method), 316
  - `_update_plot_2d` (*mlpro.bf.plot.Plottable* method), 283
  - `_update_plot_2d` (*mlpro.bf.streams.models.StreamTask* method), 326
  - `_update_plot_2d` (*mlpro.bf.streams.tasks.windows.Window* method), 447
  - `_update_plot_2d` (*mlpro.bf.systems.pool.doublependulum.DoublePendulum* method), 456
  - `_update_plot_2d` (*mlpro.oa.streams.tasks.normalizers.NormalizerMinMax* method), 428
  - `_update_plot_2d` (*mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot* method), 489
  - `_update_plot_3d` (*mlpro.bf.math.geometry.Point* method), 316
  - `_update_plot_3d` (*mlpro.bf.plot.Plottable* method), 283
  - `_update_plot_3d` (*mlpro.bf.streams.models.StreamTask* method), 326
  - `_update_plot_3d` (*mlpro.bf.streams.tasks.windows.Window* method), 447
  - `_update_plot_3d` (*mlpro.oa.streams.tasks.normalizers.NormalizerMinMax* method), 428
  - `_update_plot_nd` (*mlpro.bf.math.geometry.Point* method), 317
  - `_update_plot_nd` (*mlpro.bf.plot.Plottable* method), 283
  - `_update_plot_nd` (*mlpro.bf.streams.models.StreamTask* method), 326
  - `_update_plot_nd` (*mlpro.bf.streams.tasks.windows.Window* method), 447
  - `_update_plot_nd` (*mlpro.oa.streams.tasks.boundarydetectors.BoundaryDetector* method), 427
  - `_update_plot_nd` (*mlpro.oa.streams.tasks.normalizers.NormalizerMinMax* method), 428
  - `_utility_fct` (*mlpro.gt.dynamicgames.basics.GameBoard* method), 399
  - `_workflow` (*mlpro.oa.streams.basics.OAScenario* attribute), 423
- ## A
- `Action` (class in *mlpro.bf.systems.basics*), 341
  - `action_max` (*mlpro.rl.pool.envs.bglp.Actuator* attribute), 469, 470
  - `action_min` (*mlpro.rl.pool.envs.bglp.Actuator* attribute), 469, 470
  - `action_to_mujoco` (*mlpro.bf.systems.basics.System* method), 351
  - `ActionElement` (class in *mlpro.bf.systems.basics*), 340
  - `ActionPlanner` (class in *mlpro.rl.models\_agents*), 385
  - `actiontype` (*mlpro.rl.pool.envs.bglp.Belt* attribute), 472, 473
  - `acts` (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 478, 481
  - `Actuator` (class in *mlpro.bf.systems.basics*), 342
  - `Actuator` (class in *mlpro.rl.pool.envs.bglp*), 468
  - `adapt` (*mlpro.bf.ml.basics.Model* method), 360
  - `adapt` (*mlpro.oa.streams.basics.OATask* method), 421, 431
  - `adapt` (*mlpro.rl.models\_env\_ada.EnvModel* method), 383
  - `adapt` (*mlpro.sl.basics.SLAdaptiveFunction* method), 370
  - `adapt_on_event` (*mlpro.bf.ml.basics.Model* method), 360
  - `AdaptiveFunction` (class in *mlpro.bf.ml.basics*), 367
  - `add_action_reward` (*mlpro.rl.models\_env.Reward* method), 376
  - `add_actuator` (*mlpro.bf.systems.basics.Controller* method), 344
  - `add_agent` (*mlpro.rl.models\_agents.MultiAgent* method), 390
  - `add_agent_reward` (*mlpro.rl.models\_env.Reward* method), 376

- `add_coalition()` (*mlpro.gt.native.basics.GTCompetition* method), 413
- `add_component()` (*mlpro.bf.ui.sciui.framework.SciUIFrame* method), 289
- `add_component()` (*mlpro.bf.ui.sciui.framework.SciUISubplotRoot* method), 291
- `add_component()` (*mlpro.bf.ui.sciui.framework.SciUITabCTRL* method), 289
- `add_controller()` (*mlpro.bf.systems.basics.System* method), 349
- `add_custom_result()` (*mlpro.bf.ml.basics.TrainingResults* method), 364
- `add_dim()` (*mlpro.bf.math.basics.Set* method), 310
- `add_elem()` (*mlpro.bf.math.basics.ElementList* method), 311
- `add_element()` (*mlpro.rl.pool.sarsbuffer.PrioritizedBuffer* method), 499
- `add_element()` (*mlpro.sl.pool.afct.pytorch.PyTorchBuffer* method), 465
- `add_episode()` (*mlpro.rl.models\_train.RLDataStoring* method), 392
- `add_evaluation()` (*mlpro.rl.models\_train.RLDataStoringEval* method), 393
- `add_hp_tuple()` (*mlpro.bf.ml.basics.HyperParamDispatcher* method), 358
- `add_link_joint()` (*mlpro.rl.pool.envs.robotinhtm.RobotArm3D* method), 497
- `add_objective()` (*mlpro.bf.ml.basics.Model* method), 361
- `add_overlay()` (*mlpro.wrappers.mujoco.BaseViewer* method), 515
- `add_player()` (*mlpro.gt.dynamicgames.basics.MultiPlayer* method), 399
- `add_player()` (*mlpro.gt.native.basics.GTCoalition* method), 412
- `add_result()` (*mlpro.bf.mt.Shared* method), 298
- `add_sensor()` (*mlpro.bf.systems.basics.Controller* method), 343
- `add_system()` (*mlpro.bf.systems.basics.MultiSystem* method), 353
- `add_tab()` (*mlpro.bf.ui.sciui.framework.SciUITabCTRL* method), 289
- `add_task()` (*mlpro.bf.ml.basics.AWorkflow* method), 361
- `add_task()` (*mlpro.bf.mt.Workflow* method), 302
- `add_task()` (*mlpro.oa.streams.basics.OAWorkflow* method), 422, 433
- `add_time()` (*mlpro.bf.various.Timer* method), 275
- `add_trial()` (*mlpro.gt.native.basics.GTDataStoring* method), 415
- `AFctReward` (class in *mlpro.rl.models\_env\_ada*), 381
- `Agent` (class in *mlpro.rl.models\_agents*), 387
- `append()` (*mlpro.bf.math.basics.Set* method), 311
- `assign_so()` (*mlpro.bf.mt.Async* method), 299
- `Async` (class in *mlpro.bf.mt*), 299
- `AWorkflow` (class in *mlpro.bf.ml.basics*), 361
- ## B
- `BaseViewer` (class in *mlpro.wrappers.mujoco*), 515
- `BatchElement` (class in *mlpro.bf.math.basics*), 312
- `Belt` (class in *mlpro.rl.pool.envs.bglp*), 472
- `best_response()` (*mlpro.gt.native.basics.GTFunction* method), 406
- `best_response_value()` (*mlpro.gt.native.basics.GTPayoffMatrix* method), 407
- `BGLP` (class in *mlpro.rl.pool.envs.bglp*), 476
- `BGLP_CTRL` (class in *mlpro.gt.pool.boards.bglp*), 501
- `blts` (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 478, 481
- `BoundaryDetector` (class in *mlpro.oa.streams.tasks.boundarydetectors*), 426
- ## C
- `C_ACTION_CONSTANT` (*mlpro.bf.systems.basics.DemoScenario* attribute), 355
- `C_ACTION_RANDOM` (*mlpro.bf.systems.basics.DemoScenario* attribute), 355
- `C_ACTION_TYPE_CONT` (*mlpro.rl.pool.envs.gridworld.GridWorld* attribute), 496
- `C_ACTION_TYPE_DISC_2D` (*mlpro.rl.pool.envs.gridworld.GridWorld* attribute), 496
- `C_ANGLES_DOWN` (*mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem* attribute), 454
- `C_ANGLES_RND` (*mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem* attribute), 455
- `C_ANGLES_UP` (*mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem* attribute), 454
- `C_ANI_FRAME` (*mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem* attribute), 455
- `C_ANI_STEP` (*mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem* attribute), 455
- `C_AUTORUN_LOOP` (*mlpro.bf.mt.Task* attribute), 300
- `C_AUTORUN_NONE` (*mlpro.bf.mt.Task* attribute), 300
- `C_AUTORUN_RUN` (*mlpro.bf.mt.Task* attribute), 300
- `C_AX_CURSOR` (*mlpro.bf.ui.sciui.framework.SciUISubplot2D* attribute), 291



Index 533

<i>pro.bf.ui.sciui.framework.SciUISubplotRoot</i> attribute), 290	attribute), 436
C_FILENAME ( <i>mlpro.bf.ui.sciui.framework.SciUISubplotSaveDLG</i> attribute), 290	C_INFINITY ( <i>mlpro.rl.pool.envs.bglp.BGLP</i> attribute), 477, 480
C_FNAME_AGENT_ACTIONS ( <i>ml- pro.rl.models_train.RLTrainingResults</i> tribute), 396	C_INFINITY ( <i>mlpro.rl.pool.envs.gridworld.GridWorld</i> attribute), 496
C_FNAME_COAL_PAYOFFS ( <i>ml- pro.gt.native.basics.GTTrainingResults</i> tribute), 418	C_INFINITY ( <i>mlpro.rl.pool.envs.robotinhtm.RobotHTM</i> attribute), 498
C_FNAME_COAL_STRATEGIES ( <i>ml- pro.gt.native.basics.GTTrainingResults</i> tribute), 417	C_INST_MSG ( <i>mlpro.bf.various.Log</i> attribute), 272
C_FNAME_ENV_REWARDS ( <i>ml- pro.rl.models_train.RLTrainingResults</i> tribute), 396	C_LAP_LIMIT ( <i>mlpro.bf.various.Timer</i> attribute), 275
C_FNAME_ENV_STATES ( <i>ml- pro.rl.models_train.RLTrainingResults</i> tribute), 396	C_LATENCY ( <i>mlpro.bf.systems.basics.System</i> attribute), 348
C_FNAME_EVAL ( <i>mlpro.rl.models_train.RLTrainingResults</i> attribute), 396	C_LATENCY ( <i>mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem</i> attribute), 454
C_FONT_FAMILY ( <i>mlpro.bf.ui.sciui.framework.SciUIFramePar</i> attribute), 292	C_LATENCY ( <i>mlpro.gt.native.basics.GTGame</i> attribute), 416
C_FONT_FAMILY ( <i>mlpro.bf.ui.sciui.framework.SciUISubplotSaveDLG</i> attribute), 290	C_LATENCY ( <i>mlpro.rl.pool.envs.bglp.BGLP</i> attribute), 477, 480
C_FONT_SIZE ( <i>mlpro.bf.ui.sciui.framework.SciUIFramePar</i> attribute), 292	C_LATENCY ( <i>mlpro.rl.pool.envs.gridworld.GridWorld</i> at- tribute), 496
C_FONT_SIZE ( <i>mlpro.bf.ui.sciui.framework.SciUISubplotSaveDLG</i> attribute), 290	C_LATENCY ( <i>mlpro.rl.pool.envs.robotinhtm.RobotHTM</i> attribute), 498
C_FUNC_PAYOFF_MATRIX ( <i>ml- pro.gt.native.basics.GTFunction</i> attribute), 405	C_LOG_ALL ( <i>mlpro.bf.various.Log</i> attribute), 272
C_FUNC_TRANSFER_FCTS ( <i>ml- pro.gt.native.basics.GTFunction</i> attribute), 405	C_LOG_E ( <i>mlpro.bf.various.Log</i> attribute), 272
C_FUNCTION_TYPE ( <i>mlpro.gt.native.basics.GTFunction</i> attribute), 405	C_LOG_LEVELS ( <i>mlpro.bf.various.Log</i> attribute), 272
C_ID ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds</i> attribute), 439	C_LOG_NOthing ( <i>mlpro.bf.various.Log</i> attribute), 272
C_ID ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds</i> attribute), 440	C_LOG_SEPARATOR ( <i>mlpro.bf.ml.basics.Training</i> at- tribute), 366
C_ID ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds</i> attribute), 441	C_LOG_TYPE_E ( <i>mlpro.bf.various.Log</i> attribute), 272
C_ID ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds</i> attribute), 442	C_LOG_TYPE_I ( <i>mlpro.bf.various.Log</i> attribute), 272
C_ID ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds</i> attribute), 441	C_LOG_TYPE_S ( <i>mlpro.bf.various.Log</i> attribute), 272
C_ID ( <i>mlpro.bf.streams.streams.csv_file.StreamMLProCSV</i> attribute), 435	C_LOG_TYPE_W ( <i>mlpro.bf.various.Log</i> attribute), 272
C_ID ( <i>mlpro.bf.streams.streams.doublespiral2d.DoubleSpiral2D</i> attribute), 437	C_LOG_TYPES ( <i>mlpro.bf.various.Log</i> attribute), 272
C_ID ( <i>mlpro.bf.streams.streams.point_outliers.StreamMLProClouds</i> attribute), 438	C_LOG_WE ( <i>mlpro.bf.various.Log</i> attribute), 272
C_ID ( <i>mlpro.bf.streams.streams.rnd10d.StreamMLProRnd10D</i> attribute), 438	C_MINIMUM_VERSION ( <i>ml- pro.wrappers.mujoco.MujocoHandler</i> at- tribute), 516
	C_MODE_EVAL ( <i>mlpro.bf.ml.basics.Training</i> attribute), 366
	C_MODE_INITIAL ( <i>mlpro.bf.ops.Mode</i> attribute), 304
	C_MODE_REAL ( <i>mlpro.bf.ops.Mode</i> attribute), 304
	C_MODE_REAL ( <i>mlpro.bf.various.Timer</i> attribute), 275
	C_MODE_TRAIN ( <i>mlpro.bf.ops.Mode</i> attribute), 304
	C_MODE_TRAIN ( <i>mlpro.bf.ml.basics.Training</i> attribute), 366
	C_MODE_VIRTUAL ( <i>mlpro.bf.various.Timer</i> attribute), 275
	C_MSG_TYPE_DATA ( <i>mlpro.bf.mt.Shared</i> attribute), 298
	C_MSG_TYPE_TERM ( <i>mlpro.bf.mt.Shared</i> attribute), 298
	C_NAME ( <i>mlpro.bf.ml.basics.AdaptiveFunction</i> attribute), 367
	C_NAME ( <i>mlpro.bf.ml.basics.HyperParamTuner</i> attribute), 365
	C_NAME ( <i>mlpro.bf.ml.basics.Model</i> attribute), 359
	C_NAME ( <i>mlpro.bf.ml.basics.Scenario</i> attribute), 363
	C_NAME ( <i>mlpro.bf.ml.basics.Training</i> attribute), 366

C_NAME (mlpro.bf.mt.Workflow attribute), 302	C_NAME (mlpro.bf.various.PersonalisedStamp attribute), 277
C_NAME (mlpro.bf.ops.ScenarioBase attribute), 305	
C_NAME (mlpro.bf.physics.basics.TransferFunction attribute), 332, 333	C_NAME (mlpro.gt.dynamicgames.basics.GTTraining attribute), 400
C_NAME (mlpro.bf.physics.unitconverter.UnitConverter attribute), 336, 337	C_NAME (mlpro.gt.dynamicgames.basics.GTTrainingResults attribute), 399
C_NAME (mlpro.bf.streams.streams.clouds.StreamMLProClouds attribute), 440	C_NAME (mlpro.gt.native.basics.GTCoalition attribute), 411
C_NAME (mlpro.bf.streams.streams.clouds.StreamMLProClouds attribute), 440	C_NAME (mlpro.gt.native.basics.GTCompetition attribute), 413
C_NAME (mlpro.bf.streams.streams.clouds.StreamMLProClouds attribute), 441	C_NAME (mlpro.gt.native.basics.GTGame attribute), 416
C_NAME (mlpro.bf.streams.streams.clouds.StreamMLProClouds attribute), 442	C_NAME (mlpro.gt.native.basics.GTPlayer attribute), 410
C_NAME (mlpro.bf.streams.streams.clouds.StreamMLProClouds attribute), 441	C_NAME (mlpro.gt.native.basics.GTSolver attribute), 408
C_NAME (mlpro.bf.streams.streams.clouds.StreamMLProClouds attribute), 441	C_NAME (mlpro.gt.native.basics.GTTraining attribute), 408
C_NAME (mlpro.bf.streams.streams.csv_file.StreamMLProCSV attribute), 435	C_NAME (mlpro.gt.native.basics.GTTrainingResults attribute), 417
C_NAME (mlpro.bf.streams.streams.doublespiral2d.DoubleSpiral2D attribute), 437	C_NAME (mlpro.gt.pool.boards.bglp.BGLP_GT attribute), 501
C_NAME (mlpro.bf.streams.streams.point_outliers.StreamMLProPOutliers attribute), 438	C_NAME (mlpro.gt.pool.native.games.prisonersdilemma_2p.PrisonersDilemma attribute), 502
C_NAME (mlpro.bf.streams.streams.rnd10d.StreamMLProRnd10D attribute), 436	C_NAME (mlpro.gt.pool.native.games.prisonersdilemma_3p.PrisonersDilemma attribute), 503
C_NAME (mlpro.bf.streams.tasks.deriver.Deriver attribute), 443	C_NAME (mlpro.gt.pool.native.games.rockpaperscissors.RockPaperScissors attribute), 504
C_NAME (mlpro.bf.streams.tasks.rearranger.Rearranger attribute), 444	C_NAME (mlpro.gt.pool.native.games.routingproblems_3p.Routing_3P attribute), 508
C_NAME (mlpro.bf.streams.tasks.windows.Window attribute), 445	C_NAME (mlpro.gt.pool.native.games.supplydemand_3p.SupplyDemand_3P attribute), 510
C_NAME (mlpro.bf.systems.basics.DemoScenario attribute), 355	C_NAME (mlpro.gt.pool.native.solvers.greedypolicy.MaxGreedyPolicy attribute), 511
C_NAME (mlpro.bf.systems.basics.SystemShared attribute), 345	C_NAME (mlpro.gt.pool.native.solvers.greedypolicy.MinGreedyPolicy attribute), 511
C_NAME (mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem attribute), 454	C_NAME (mlpro.gt.pool.native.solvers.randomsolver.RandomSolver attribute), 512
C_NAME (mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem attribute), 458	C_NAME (mlpro.oa.streams.basics.OATraining attribute), 425, 434
C_NAME (mlpro.bf.systems.pool.doublependulum.DoublePendulumSystem attribute), 460	C_NAME (mlpro.oa.streams.tasks.boundarydetectors.BoundaryDetector attribute), 426
C_NAME (mlpro.bf.ui.sciui.framework.SciUIComponent attribute), 288	C_NAME (mlpro.oa.streams.tasks.normalizers.NormalizerMinMax attribute), 428
C_NAME (mlpro.bf.ui.sciui.framework.SciUIFrame attribute), 289	C_NAME (mlpro.oa.streams.tasks.normalizers.NormalizerZTransform attribute), 429
C_NAME (mlpro.bf.ui.sciui.framework.SciUIFrameParam attribute), 292	C_NAME (mlpro.rl.models_agents.Agent attribute), 388
C_NAME (mlpro.bf.ui.sciui.framework.SciUIScenario attribute), 292	C_NAME (mlpro.rl.models_agents.MultiAgent attribute), 389
C_NAME (mlpro.bf.ui.sciui.framework.SciUISubplotSaveDLG attribute), 290	C_NAME (mlpro.rl.models_agents.Policy attribute), 385
C_NAME (mlpro.bf.ui.sciui.framework.SciUIWindow attribute), 288	C_NAME (mlpro.rl.models_agents.RLScenarioMBInt attribute), 387
C_NAME (mlpro.bf.ui.sciui.main.SciUI attribute), 285	C_NAME (mlpro.rl.models_env_ada.EnvModel attribute), 382
C_NAME (mlpro.bf.various.Log attribute), 272	C_NAME (mlpro.rl.models_train.RLScenario attribute), 393, 394
	C_NAME (mlpro.rl.models_train.RLTraining attribute), 393

397		attribute), 273
C_NAME (mlpro.rl.models.train.RLTrainingResults attribute), 396	C_PLAYER_FOLLOWER (ml-pro.gt.dynamicgames.stackelberg.GTPlayer_SG attribute), 401	
C_NAME (mlpro.rl.pool.envs.bglp.BGLP attribute), 477, 480	C_PLAYER_LEADER (ml-pro.gt.dynamicgames.stackelberg.GTPlayer_SG attribute), 401	
C_NAME (mlpro.rl.pool.envs.doublependulum.DoublePendulumOA4 attribute), 494	C_PLOT_ACTIVE (mlpro.bf.math.geometry.Point attribute), 316	
C_NAME (mlpro.rl.pool.envs.doublependulum.DoublePendulumOA5 attribute), 495	C_PLOT_ACTIVE (mlpro.bf.mt.Workflow attribute), 302	
C_NAME (mlpro.rl.pool.envs.doublependulum.DoublePendulumOA7 attribute), 491	C_PLOT_ACTIVE (mlpro.bf.plot.Plottable attribute), 280, 281	
C_NAME (mlpro.rl.pool.envs.doublependulum.DoublePendulumS7 attribute), 493	C_PLOT_ACTIVE (mlpro.bf.streams.models.StreamScenario attribute), 328	
C_NAME (mlpro.rl.pool.envs.gridworld.GridWorld attribute), 496	C_PLOT_ACTIVE (mlpro.bf.streams.models.StreamTask attribute), 325	
C_NAME (mlpro.rl.pool.envs.robotinhtm.RobotHTM attribute), 498	C_PLOT_ACTIVE (mlpro.bf.streams.models.StreamWorkflow attribute), 327	
C_NAME (mlpro.sl.basics.SLAdaptiveFunction attribute), 370	C_PLOT_ACTIVE (mlpro.bf.systems.basics.System attribute), 348	
C_NAME (mlpro.wrappers.mujoco.MujocoHandler attribute), 516	C_PLOT_ACTIVE (mlpro.bf.systems.pool.doublependulum.DoublePendulum attribute), 454	
C_NUM_DIMENSIONS (ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dynamic attribute), 440	C_PLOT_ACTIVE (mlpro.oa.streams.basics.OATask attribute), 421, 431	
C_NUM_DIMENSIONS (ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dynamic attribute), 441	C_PLOT_ACTIVE (mlpro.rl.pool.envs.doublependulum.DoublePendulumOA7 attribute), 495	
C_NUM_DIMENSIONS (ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dynamic attribute), 442	C_PLOT_DEFAULT_VIEW (mlpro.bf.plot.Plottable attribute), 281	
C_NUM_DIMENSIONS (ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dynamic attribute), 441	C_PLOT_DEFAULT_VIEW (mlpro.bf.streams.models.StreamTask attribute), 325	
C_NUM_DIMENSIONS (ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dynamic attribute), 441	C_PLOT_DEFAULT_VIEW (mlpro.bf.systems.pool.doublependulum.DoublePendulumSystemRoom attribute), 454	
C_NUM_INSTANCES (ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dynamic attribute), 440	C_PLOT_DEFAULT_VIEW (mlpro.oa.streams.basics.OATask attribute), 421, 431	
C_NUM_INSTANCES (ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dynamic attribute), 441	C_PLOT_DEFAULT_VIEW (mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot attribute), 487	
C_NUM_INSTANCES (ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dynamic attribute), 442	C_PLOT_DEFAULT_VIEW (mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot attribute), 487	
C_NUM_INSTANCES (ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dynamic attribute), 441	C_PLOT_DEFAULT_VIEW (mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot attribute), 487	
C_NUM_INSTANCES (ml-pro.bf.streams.streams.doublespiral2d.DoubleSpiral2D attribute), 437	C_PLOT_IN_WINDOW (ml-pro.bf.streams.tasks.windows.Window attribute), 445	
C_NUM_INSTANCES (ml-pro.bf.streams.streams.rnd10d.StreamMLProRnd10D attribute), 436	C_PLOT_ND_XLABEL_FEATURE (ml-pro.oa.streams.tasks.boundarydetectors.BoundaryDetector attribute), 426	
C_NUMERIC_BASE_SETS (mlpro.bf.math.basics.Set attribute), 310	C_PLOT_ND_XLABEL_INST (ml-pro.bf.streams.models.StreamTask attribute),	
C_PERSISTENCE_VERSION (mlpro.bf.various.Persistent attribute),		



<a href="#">325</a>		<a href="#">C_REWARD_TYPE</a> ( <i>mlpro.rl.models.env.EnvBase</i> attribute), <a href="#">378</a>
<a href="#">C_PLOT_ND_XLABEL_TIME</a> ( <i>ml-pro.bf.streams.models.StreamTask</i> attribute), <a href="#">325</a>		<a href="#">C_REWARD_TYPE</a> ( <i>mlpro.rl.pool.envs.bglp.BGLP</i> attribute), <a href="#">477</a> , <a href="#">480</a>
<a href="#">C_PLOT_ND_YLABEL</a> ( <i>ml-pro.bf.streams.models.StreamTask</i> attribute), <a href="#">325</a>		<a href="#">C_REWARD_TYPE</a> ( <i>mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot</i> attribute), <a href="#">487</a>
<a href="#">C_PLOT_ND_YLABEL</a> ( <i>ml-pro.oa.streams.tasks.boundarydetectors.BoundaryDetector</i> attribute), <a href="#">426</a>		<a href="#">C_REWARD_TYPE</a> ( <i>mlpro.rl.pool.envs.gridworld.GridWorld</i> attribute), <a href="#">496</a>
<a href="#">C_PLOT_OUTSIDE_WINDOW</a> ( <i>ml-pro.bf.streams.tasks.windows.Window</i> attribute), <a href="#">445</a>		<a href="#">C_REWARD_TYPE</a> ( <i>mlpro.rl.pool.envs.robotinhtml.RobotHTML</i> attribute), <a href="#">498</a>
<a href="#">C_PLOT_STANDALONE</a> ( <i>mlpro.bf.plot.Plottable</i> attribute), <a href="#">280</a> , <a href="#">281</a>		<a href="#">C_RST_BALANCING_001</a> ( <i>ml-pro.rl.pool.envs.doublependulum.DoublePendulumRoot</i> attribute), <a href="#">488</a>
<a href="#">C_PLOT_STANDALONE</a> ( <i>ml-pro.bf.streams.models.StreamTask</i> attribute), <a href="#">325</a>		<a href="#">C_RST_BALANCING_002</a> ( <i>ml-pro.rl.pool.envs.doublependulum.DoublePendulumRoot</i> attribute), <a href="#">488</a>
<a href="#">C_PLOT_STANDALONE</a> ( <i>ml-pro.bf.streams.tasks.deriver.Deriver</i> attribute), <a href="#">443</a>		<a href="#">C_RST_SWINGING_001</a> ( <i>ml-pro.rl.pool.envs.doublependulum.DoublePendulumRoot</i> attribute), <a href="#">488</a>
<a href="#">C_PLOT_STANDALONE</a> ( <i>ml-pro.bf.streams.tasks.rearranger.Rearranger</i> attribute), <a href="#">444</a>		<a href="#">C_RST_SWINGING_OUTER_POLE_001</a> ( <i>ml-pro.rl.pool.envs.doublependulum.DoublePendulumRoot</i> attribute), <a href="#">488</a>
<a href="#">C_PLOT_STANDALONE</a> ( <i>ml-pro.bf.streams.tasks.windows.Window</i> attribute), <a href="#">445</a>		<a href="#">C_SCIREF_ABSTRACT</a> ( <i>ml-pro.bf.streams.streams.clouds.StreamMLProClouds</i> attribute), <a href="#">440</a>
<a href="#">C_PLOT_STANDALONE</a> ( <i>mlpro.oa.streams.basics.OATask</i> attribute), <a href="#">421</a> , <a href="#">431</a>		<a href="#">C_SCIREF_ABSTRACT</a> ( <i>ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C1000State</i> attribute), <a href="#">440</a>
<a href="#">C_PLOT_TYPE_CY</a> ( <i>mlpro.bf.plot.DataPlotting</i> attribute), <a href="#">284</a>		<a href="#">C_SCIREF_ABSTRACT</a> ( <i>ml-pro.bf.streams.streams.clouds.StreamMLProClouds2D4C5000Dy</i> attribute), <a href="#">442</a>
<a href="#">C_PLOT_TYPE_EP</a> ( <i>mlpro.bf.plot.DataPlotting</i> attribute), <a href="#">284</a>		<a href="#">C_SCIREF_ABSTRACT</a> ( <i>ml-pro.bf.streams.streams.clouds.StreamMLProClouds3D8C1000D</i> attribute), <a href="#">442</a>
<a href="#">C_PLOT_TYPE_EP_M</a> ( <i>mlpro.bf.plot.DataPlotting</i> attribute), <a href="#">284</a>		<a href="#">C_SCIREF_ABSTRACT</a> ( <i>ml-pro.bf.streams.streams.clouds.StreamMLProClouds3D8C2000State</i> attribute), <a href="#">441</a>
<a href="#">C_PLOT_VALID_VIEWS</a> ( <i>mlpro.bf.plot.Plottable</i> attribute), <a href="#">281</a>		<a href="#">C_SCIREF_ABSTRACT</a> ( <i>ml-pro.bf.streams.streams.doublespiral2d.DoubleSpiral2D</i> attribute), <a href="#">437</a>
<a href="#">C_PLOT_VALID_VIEWS</a> ( <i>ml-pro.bf.streams.models.StreamTask</i> attribute), <a href="#">325</a>		<a href="#">C_SCIREF_ABSTRACT</a> ( <i>ml-pro.bf.streams.streams.point_outliers.StreamMLProPOutliers</i> attribute), <a href="#">438</a>
<a href="#">C_PLOT_VALID_VIEWS</a> ( <i>mlpro.oa.streams.basics.OATask</i> attribute), <a href="#">421</a> , <a href="#">431</a>		<a href="#">C_SCIREF_ABSTRACT</a> ( <i>ml-pro.bf.streams.streams.rnd10d.StreamMLProRnd10D</i> attribute), <a href="#">436</a>
<a href="#">C_PLOT_VALID_VIEWS</a> ( <i>ml-pro.oa.streams.tasks.boundarydetectors.BoundaryDetector</i> attribute), <a href="#">426</a>		<a href="#">C_SCIREF_ABSTRACT</a> ( <i>mlpro.bf.various.ScientificObject</i> attribute), <a href="#">276</a>
<a href="#">C_PROPERTY_POS</a> ( <i>mlpro.bf.math.geometry.Point</i> attribute), <a href="#">316</a>		<a href="#">C_SCIREF_ADDRESS</a> ( <i>mlpro.bf.various.ScientificObject</i> attribute), <a href="#">277</a>
<a href="#">C_RANGE_NONE</a> ( <i>mlpro.bf.mt.Range</i> attribute), <a href="#">297</a>		<a href="#">C_SCIREF_AUTHOR</a> ( <i>ml-pro.bf.streams.samplers.min_wise.SamplerMinWise</i> attribute), <a href="#">448</a>
<a href="#">C_RANGE_PROCESS</a> ( <i>mlpro.bf.mt.Range</i> attribute), <a href="#">297</a>		<a href="#">C_SCIREF_AUTHOR</a> ( <i>ml-</i>
<a href="#">C_RANGE_THREAD</a> ( <i>mlpro.bf.mt.Range</i> attribute), <a href="#">297</a>		
<a href="#">C_RELEASED</a> ( <i>mlpro.bf.ui.sciui.framework.SciUIScenario</i> attribute), <a href="#">292</a>		
<a href="#">C_REWARD_TYPE</a> ( <i>mlpro.gt.dynamicgames.basics.GameBoard</i> attribute), <a href="#">399</a>		

<i>pro.bf.streams.samplers.reservoir_sampling.SamplerReservoir</i> (ml- attribute), 449	<i>C_SCIREF_MONTH</i> ( <i>mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir</i> attribute), 450
<i>C_SCIREF_AUTHOR</i> ( <i>mlpro.bf.streams.streams.csv_file.StreamMLProCSV</i> attribute), 435	<i>C_SCIREF_MONTH</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276
<i>C_SCIREF_AUTHOR</i> ( <i>mlpro.bf.streams.streams.csv_file.StreamMLProCSV</i> attribute), 435	<i>C_SCIREF_NOTES</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 277
<i>C_SCIREF_AUTHOR</i> ( <i>mlpro.bf.streams.streams.csv_file.StreamMLProCSV</i> attribute), 435	<i>C_SCIREF_NUMBER</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276
<i>C_SCIREF_AUTHOR</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276	<i>C_SCIREF_NUMBER</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276
<i>C_SCIREF_BOOKTITLE</i> ( <i>mlpro.bf.streams.samplers.min_wise.SamplerMinWise</i> attribute), 448	<i>C_SCIREF_NUMPAGES</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 277
<i>C_SCIREF_BOOKTITLE</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 277	<i>C_SCIREF_PAGES</i> ( <i>mlpro.bf.streams.samplers.min_wise.SamplerMinWise</i> attribute), 448
<i>C_SCIREF_CHAPTER</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 277	<i>C_SCIREF_PAGES</i> ( <i>mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir</i> attribute), 450
<i>C_SCIREF_CITY</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276	<i>C_SCIREF_PAGES</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276
<i>C_SCIREF_CONFERENCE</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 277	<i>C_SCIREF_PUBLISHER</i> ( <i>mlpro.bf.streams.samplers.min_wise.SamplerMinWise</i> attribute), 448
<i>C_SCIREF_COUNTRY</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276	<i>C_SCIREF_PUBLISHER</i> ( <i>mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir</i> attribute), 450
<i>C_SCIREF_DAY</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276	<i>C_SCIREF_PUBLISHER</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276
<i>C_SCIREF_DOI</i> ( <i>mlpro.bf.streams.samplers.min_wise.SamplerMinWise</i> attribute), 448	<i>C_SCIREF_SERIES</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276
<i>C_SCIREF_DOI</i> ( <i>mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir</i> attribute), 450	<i>C_SCIREF_TITLE</i> ( <i>mlpro.bf.streams.samplers.min_wise.SamplerMinWise</i> attribute), 448
<i>C_SCIREF_DOI</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276	<i>C_SCIREF_TITLE</i> ( <i>mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir</i> attribute), 449
<i>C_SCIREF_EDITOR</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 277	<i>C_SCIREF_TITLE</i> ( <i>mlpro.bf.systems.pool.doublependulum.DoublePendulum</i> attribute), 454
<i>C_SCIREF_HOWPUBLISHED</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 277	<i>C_SCIREF_TITLE</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276
<i>C_SCIREF_INSTITUTION</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 277	<i>C_SCIREF_TYPE</i> ( <i>mlpro.bf.ml.basics.Model</i> attribute), 359
<i>C_SCIREF_ISBN</i> ( <i>mlpro.bf.streams.samplers.min_wise.SamplerMinWise</i> attribute), 448	<i>C_SCIREF_TYPE</i> ( <i>mlpro.bf.streams.samplers.min_wise.SamplerMinWise</i> attribute), 448
<i>C_SCIREF_ISBN</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276	<i>C_SCIREF_TYPE</i> ( <i>mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir</i> attribute), 449
<i>C_SCIREF_ISSN</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 277	<i>C_SCIREF_TYPE</i> ( <i>mlpro.bf.streams.streams.csv_file.StreamMLProCSV</i> attribute), 435
<i>C_SCIREF_JOURNAL</i> ( <i>mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir</i> attribute), 450	<i>C_SCIREF_TYPE</i> ( <i>mlpro.bf.systems.pool.doublependulum.DoublePendulum</i> attribute), 454
<i>C_SCIREF_JOURNAL</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276	<i>C_SCIREF_TYPE</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276
<i>C_SCIREF_KEYWORDS</i> ( <i>mlpro.bf.various.ScientificObject</i> attribute), 276	<i>C_SCIREF_TYPE</i> ( <i>mlpro.gt.native.basics.GTSolver</i> attribute), 408
	<i>C_SCIREF_TYPE_ARTICLE</i> ( <i>mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir</i> attribute), 450

<i>attribute</i> ), 449		<i>attribute</i> ), 449	
C_SCIREF_TYPE_ARTICLE	(mlpro.bf.various.ScientificObject attribute), 276	C_SCIREF_YEAR	(mlpro.bf.various.ScientificObject attribute), 276
C_SCIREF_TYPE_BOOK	(mlpro.bf.various.ScientificObject attribute), 276	C_SUFFIX	(mlpro.bf.various.Persistent attribute), 273
C_SCIREF_TYPE_BOOKLET	(mlpro.bf.various.ScientificObject attribute), 276	C_SUFFIXES	(mlpro.bf.ui.sciui.framework.SciUISubplotSaveDLG attribute), 290
C_SCIREF_TYPE_INBOOK	(mlpro.bf.various.ScientificObject attribute), 276	C_THRSH_GOAL	(mlpro.bf.systems.pool.doublependulum.DoublePendulumS attribute), 455
C_SCIREF_TYPE_INPROCEEDINGS	(mlpro.bf.various.ScientificObject attribute), 276	C_TRF_FUNC_APPROX	(mlpro.bf.physics.basics.TransferFunction attribute), 332, 333
C_SCIREF_TYPE_NONE	(mlpro.bf.various.ScientificObject attribute), 276	C_TRF_FUNC_CUSTOM	(mlpro.bf.physics.basics.TransferFunction attribute), 332
C_SCIREF_TYPE_ONLINE	(mlpro.bf.various.ScientificObject attribute), 276	C_TRF_FUNC_LINEAR	(mlpro.bf.physics.basics.TransferFunction attribute), 332
C_SCIREF_TYPE_PROCEEDINGS	(mlpro.bf.streams.samplers.min_wise.SamplerMinWise attribute), 448	C_TYPE	(mlpro.bf.events.EventManager attribute), 294
C_SCIREF_TYPE_PROCEEDINGS	(mlpro.bf.various.ScientificObject attribute), 276	C_TYPE	(mlpro.bf.math.basics.Dimension attribute), 309
C_SCIREF_TYPE_TECHREPORT	(mlpro.bf.various.ScientificObject attribute), 276	C_TYPE	(mlpro.bf.ml.basics.AdaptiveFunction attribute), 367
C_SCIREF_TYPE_UNPUBLISHED	(mlpro.bf.various.ScientificObject attribute), 276	C_TYPE	(mlpro.bf.ml.basics.AWorkflow attribute), 361
C_SCIREF_URL (mlpro.bf.streams.samplers.min_wise.SamplerMinWise attribute), 448		C_TYPE	(mlpro.bf.ml.basics.HyperParamTuner attribute), 365
C_SCIREF_URL (mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir attribute), 450		C_TYPE	(mlpro.bf.ml.basics.Model attribute), 359
C_SCIREF_URL (mlpro.bf.streams.streams.csv_file.StreamMCPCSV attribute), 435		C_TYPE	(mlpro.bf.ml.basics.Scenario attribute), 363
C_SCIREF_URL (mlpro.bf.systems.pool.doublependulum.DoublePendulumSystemRoot attribute), 454		C_TYPE	(mlpro.bf.ml.basics.Training attribute), 366
C_SCIREF_URL (mlpro.bf.various.ScientificObject attribute), 276		C_TYPE	(mlpro.bf.ml.basics.TrainingResults attribute), 364
C_SCIREF_VERSION (mlpro.bf.various.ScientificObject attribute), 277		C_TYPE	(mlpro.bf.mt.Task attribute), 300
C_SCIREF_VOLUME	(mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir attribute), 450	C_TYPE	(mlpro.bf.mt.Workflow attribute), 302
C_SCIREF_VOLUME (mlpro.bf.various.ScientificObject attribute), 276		C_TYPE	(mlpro.bf.ops.ScenarioBase attribute), 305
C_SCIREF_YEAR (mlpro.bf.streams.samplers.min_wise.SamplerMinWise attribute), 448		C_TYPE	(mlpro.bf.physics.basics.TransferFunction attribute), 332
C_SCIREF_YEAR (mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir attribute), 449		C_TYPE	(mlpro.bf.physics.unitconverter.UnitConverter attribute), 336, 337
C_SCIREF_YEAR (mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir attribute), 449		C_TYPE	(mlpro.bf.streams.models.Instance attribute), 319
		C_TYPE	(mlpro.bf.streams.models.Sampler attribute), 320
		C_TYPE	(mlpro.bf.streams.models.Stream attribute), 321
		C_TYPE	(mlpro.bf.streams.models.StreamProvider attribute), 322
		C_TYPE	(mlpro.bf.streams.models.StreamScenario attribute), 328
		C_TYPE	(mlpro.bf.streams.models.StreamTask attribute), 325
		C_TYPE	(mlpro.bf.streams.models.StreamWorkflow attribute), 327
		C_TYPE	(mlpro.bf.streams.samplers.min_wise.SamplerMinWise attribute), 448
		C_TYPE	(mlpro.bf.streams.samplers.random.SamplerRND attribute), 449
		C_TYPE	(mlpro.bf.streams.samplers.reservoir_sampling.SamplerReservoir attribute), 449
		C_TYPE	(mlpro.bf.streams.samplers.weighted_random.SamplerWeightedRND attribute), 449

<i>attribute</i> ), 450	<i>C_TYPE</i> ( <i>mlpro.gt.native.basics.GTCoalition</i> <i>attribute</i> ), 411
<i>C_TYPE</i> ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds</i> <i>attribute</i> ), 440	<i>C_TYPE</i> ( <i>mlpro.gt.native.basics.GTCompetition</i> <i>attribute</i> ), 413
<i>C_TYPE</i> ( <i>mlpro.bf.streams.streams.csv_file.StreamMLProCSV</i> <i>attribute</i> ), 435	<i>C_TYPE</i> ( <i>mlpro.gt.native.basics.GTFunction</i> <i>attribute</i> ), 405
<i>C_TYPE</i> ( <i>mlpro.bf.streams.streams.point_outliers.StreamMLProPOutliers</i> <i>attribute</i> ), 438	<i>C_TYPE</i> ( <i>mlpro.gt.native.basics.GTGame</i> <i>attribute</i> ), 416
<i>C_TYPE</i> ( <i>mlpro.bf.systems.basics.Actuator</i> <i>attribute</i> ), 342	<i>C_TYPE</i> ( <i>mlpro.gt.native.basics.GTPayoffMatrix</i> <i>attribute</i> ), 407
<i>C_TYPE</i> ( <i>mlpro.bf.systems.basics.Controller</i> <i>attribute</i> ), 343	<i>C_TYPE</i> ( <i>mlpro.gt.native.basics.GTPlayer</i> <i>attribute</i> ), 410
<i>C_TYPE</i> ( <i>mlpro.bf.systems.basics.FctBroken</i> <i>attribute</i> ), 342	<i>C_TYPE</i> ( <i>mlpro.gt.native.basics.GTSolver</i> <i>attribute</i> ), 408
<i>C_TYPE</i> ( <i>mlpro.bf.systems.basics.FctSTrans</i> <i>attribute</i> ), 341	<i>C_TYPE</i> ( <i>mlpro.gt.native.basics.GTStrategy</i> <i>attribute</i> ), 405
<i>C_TYPE</i> ( <i>mlpro.bf.systems.basics.FctSuccess</i> <i>attribute</i> ), 342	<i>C_TYPE</i> ( <i>mlpro.gt.native.basics.GTTraining</i> <i>attribute</i> ), 418
<i>C_TYPE</i> ( <i>mlpro.bf.systems.basics.MultiSystem</i> <i>attribute</i> ), 353	<i>C_TYPE</i> ( <i>mlpro.oa.streams.basics.OAScenario</i> <i>attribute</i> ), 423, 433
<i>C_TYPE</i> ( <i>mlpro.bf.systems.basics.Sensor</i> <i>attribute</i> ), 342	<i>C_TYPE</i> ( <i>mlpro.oa.streams.basics.OATask</i> <i>attribute</i> ), 421, 431
<i>C_TYPE</i> ( <i>mlpro.bf.systems.basics.System</i> <i>attribute</i> ), 348	<i>C_TYPE</i> ( <i>mlpro.oa.streams.basics.OAWorkflow</i> <i>attribute</i> ), 422, 433
<i>C_TYPE</i> ( <i>mlpro.bf.ui.sciui.framework.SciUIComponent</i> <i>attribute</i> ), 288	<i>C_TYPE</i> ( <i>mlpro.rl.models_agents.ActionPlanner</i> <i>attribute</i> ), 386
<i>C_TYPE</i> ( <i>mlpro.bf.ui.sciui.framework.SciUIFrame</i> <i>attribute</i> ), 289	<i>C_TYPE</i> ( <i>mlpro.rl.models_agents.Agent</i> <i>attribute</i> ), 388
<i>C_TYPE</i> ( <i>mlpro.bf.ui.sciui.framework.SciUIFrameParam</i> <i>attribute</i> ), 292	<i>C_TYPE</i> ( <i>mlpro.rl.models_agents.MultiAgent</i> <i>attribute</i> ), 389
<i>C_TYPE</i> ( <i>mlpro.bf.ui.sciui.framework.SciUIScenario</i> <i>attribute</i> ), 292	<i>C_TYPE</i> ( <i>mlpro.rl.models_agents.Policy</i> <i>attribute</i> ), 385
<i>C_TYPE</i> ( <i>mlpro.bf.ui.sciui.framework.SciUISubplot2D</i> <i>attribute</i> ), 291	<i>C_TYPE</i> ( <i>mlpro.rl.models_env.EnvBase</i> <i>attribute</i> ), 378
<i>C_TYPE</i> ( <i>mlpro.bf.ui.sciui.framework.SciUISubplot3D</i> <i>attribute</i> ), 291	<i>C_TYPE</i> ( <i>mlpro.rl.models_env.Environment</i> <i>attribute</i> ), 380
<i>C_TYPE</i> ( <i>mlpro.bf.ui.sciui.framework.SciUISubplotRoot</i> <i>attribute</i> ), 290	<i>C_TYPE</i> ( <i>mlpro.rl.models_env.FctReward</i> <i>attribute</i> ), 376
<i>C_TYPE</i> ( <i>mlpro.bf.ui.sciui.framework.SciUITabCTRL</i> <i>attribute</i> ), 289	<i>C_TYPE</i> ( <i>mlpro.rl.models_env_ada.AFctReward</i> <i>attribute</i> ), 381
<i>C_TYPE</i> ( <i>mlpro.bf.ui.sciui.framework.SciUIWindow</i> <i>attribute</i> ), 287	<i>C_TYPE</i> ( <i>mlpro.rl.models_env_ada.EnvModel</i> <i>attribute</i> ), 382
<i>C_TYPE</i> ( <i>mlpro.bf.ui.sciui.main.SciUI</i> <i>attribute</i> ), 285	<i>C_TYPE</i> ( <i>mlpro.rl.models_train.RLScenario</i> <i>attribute</i> ), 393, 394
<i>C_TYPE</i> ( <i>mlpro.bf.various.Log</i> <i>attribute</i> ), 272	<i>C_TYPE</i> ( <i>mlpro.rl.pool.actionplanner.mpc.MPC</i> <i>attribute</i> ), 467
<i>C_TYPE</i> ( <i>mlpro.gt.dynamicgames.basics.Game</i> <i>attribute</i> ), 399	<i>C_TYPE</i> ( <i>mlpro.sl.basics.SLAdaptiveFunction</i> <i>attribute</i> ), 370
<i>C_TYPE</i> ( <i>mlpro.gt.dynamicgames.basics.GameBoard</i> <i>attribute</i> ), 399	<i>C_TYPE</i> ( <i>mlpro.sl.fnn.FNN</i> <i>attribute</i> ), 372
<i>C_TYPE</i> ( <i>mlpro.gt.dynamicgames.basics.MultiPlayer</i> <i>attribute</i> ), 399	<i>C_TYPE</i> ( <i>mlpro.sl.fnn.MLP</i> <i>attribute</i> ), 373
<i>C_TYPE</i> ( <i>mlpro.gt.dynamicgames.basics.Player</i> <i>attribute</i> ), 399	<i>C_TYPE</i> ( <i>mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP</i> <i>attribute</i> ), 462
<i>C_TYPE</i> ( <i>mlpro.gt.dynamicgames.potential.PGameBoard</i> <i>attribute</i> ), 401	<i>C_TYPE</i> ( <i>mlpro.wrappers.mujoco.MujocoHandler</i> <i>attribute</i> ), 516
<i>C_TYPE</i> ( <i>mlpro.gt.dynamicgames.stackelberg.GTMultiPlayer_SG</i> <i>attribute</i> ), 402	<i>C_TYPE EVERY ACTION</i> ( <i>mlpro.rl.models_env.Reward</i> <i>attribute</i> ), 376
<i>C_TYPE</i> ( <i>mlpro.gt.dynamicgames.stackelberg.GTPlayer_SG</i> <i>attribute</i> ), 401	<i>C_TYPE EVERY AGENT</i> ( <i>mlpro.rl.models_env.Reward</i> <i>attribute</i> ), 376
	<i>C_TYPE OVERALL</i> ( <i>mlpro.rl.models_env.Reward</i> <i>attribute</i> ), 376
	<i>C_UNIT_CONV_CURRENT</i> ( <i>ml-</i>



<i>pro.bf.physics.unitconverter.UnitConverter</i> <i>attribute</i> ), 336, 337	C_VAR_DAY ( <i>mlpro.rl.models_train.RLDataStoring</i> <i>attribute</i> ), 392
C_UNIT_CONV_FORCE ( <i>ml-</i> <i>pro.bf.physics.unitconverter.UnitConverter</i> <i>attribute</i> ), 336, 337	C_VAR_MICROSEC ( <i>mlpro.gt.native.basics.GTDataStoring</i> <i>attribute</i> ), 415
C_UNIT_CONV_LENGTH ( <i>ml-</i> <i>pro.bf.physics.unitconverter.UnitConverter</i> <i>attribute</i> ), 336, 337	C_VAR_MICROSEC ( <i>mlpro.rl.models_train.RLDataStoring</i> <i>attribute</i> ), 392
C_UNIT_CONV_MASS ( <i>ml-</i> <i>pro.bf.physics.unitconverter.UnitConverter</i> <i>attribute</i> ), 337	C_VAR_NUM_ADAPT ( <i>ml-</i> <i>pro.rl.models_train.RLDataStoringEval</i> <i>at-</i> <i>tribute</i> ), 392
C_UNIT_CONV_POWER ( <i>ml-</i> <i>pro.bf.physics.unitconverter.UnitConverter</i> <i>attribute</i> ), 336, 337	C_VAR_NUM_BROKEN ( <i>ml-</i> <i>pro.rl.models_train.RLDataStoringEval</i> <i>at-</i> <i>tribute</i> ), 392
C_UNIT_CONV_PRESSURE ( <i>ml-</i> <i>pro.bf.physics.unitconverter.UnitConverter</i> <i>attribute</i> ), 336, 337	C_VAR_NUM_CYCLES ( <i>ml-</i> <i>pro.rl.models_train.RLDataStoringEval</i> <i>at-</i> <i>tribute</i> ), 392
C_UNIT_CONV_TEMPERATURE ( <i>ml-</i> <i>pro.bf.physics.unitconverter.UnitConverter</i> <i>attribute</i> ), 337	C_VAR_NUM_LIMIT ( <i>ml-</i> <i>pro.rl.models_train.RLDataStoringEval</i> <i>at-</i> <i>tribute</i> ), 392
C_UNIT_CONV_TIME ( <i>ml-</i> <i>pro.bf.physics.unitconverter.UnitConverter</i> <i>attribute</i> ), 337	C_VAR_NUM_SUCCESS ( <i>ml-</i> <i>pro.rl.models_train.RLDataStoringEval</i> <i>at-</i> <i>tribute</i> ), 392
C_VALID_ANGLES ( <i>mlpro.bf.systems.pool.doublependulum.DoublePendulumRoot</i> <i>attribute</i> ), 455	C_VAR_SCORE ( <i>mlpro.bf.ml.basics.HyperParamTuner</i> <i>at-</i> <i>tribute</i> ), 365
C_VALID_DEPTH ( <i>mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot</i> <i>attribute</i> ), 488	C_VAR_SCORE_MA ( <i>mlpro.rl.models_train.RLDataStoringEval</i> <i>attribute</i> ), 392
C_VALID_MODES ( <i>mlpro.bf.ops.Mode</i> <i>attribute</i> ), 304	C_VAR_SCORE_MA_UNTIL_STAG ( <i>ml-</i> <i>pro.rl.models_train.RLDataStoringEval</i> <i>at-</i> <i>tribute</i> ), 392
C_VALID_RANGES ( <i>mlpro.bf.mt.Range</i> <i>attribute</i> ), 297	C_VAR_SCORE_UNTIL_STAG ( <i>ml-</i> <i>pro.rl.models_train.RLDataStoringEval</i> <i>at-</i> <i>tribute</i> ), 392
C_VALID_RST_BALANCING ( <i>ml-</i> <i>pro.rl.pool.envs.doublependulum.DoublePendulumRoot</i> <i>attribute</i> ), 488	C_VAR_SEC ( <i>mlpro.gt.native.basics.GTDataStoring</i> <i>at-</i> <i>tribute</i> ), 415
C_VALID_RST_SWINGING ( <i>ml-</i> <i>pro.rl.pool.envs.doublependulum.DoublePendulumRoot</i> <i>attribute</i> ), 488	C_VAR_SEC ( <i>mlpro.rl.models_train.RLDataStoring</i> <i>attribute</i> ), 392
C_VALID_RST_SWINGING_OUTER_POLE ( <i>ml-</i> <i>pro.rl.pool.envs.doublependulum.DoublePendulumRoot</i> <i>attribute</i> ), 488	C_VAR_TRIAL ( <i>mlpro.bf.ml.basics.HyperParamTuner</i> <i>at-</i> <i>tribute</i> ), 365
C_VALID_TYPES ( <i>mlpro.rl.models_env.Reward</i> <i>attribute</i> ), 376	C_VERSION ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds</i> <i>attribute</i> ), 440
C_VALID_VIEWS ( <i>mlpro.bf.plot.PlotSettings</i> <i>attribute</i> ), 280	C_VERSION ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds2D4C10</i> <i>attribute</i> ), 440
C_VAR0 ( <i>mlpro.gt.native.basics.GTDataStoring</i> <i>at-</i> <i>tribute</i> ), 415	C_VERSION ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds2D4C50</i> <i>attribute</i> ), 441
C_VAR0 ( <i>mlpro.rl.models_train.RLDataStoring</i> <i>attribute</i> ), 392	C_VERSION ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds3D8C10</i> <i>attribute</i> ), 442
C_VAR0 ( <i>mlpro.rl.models_train.RLDataStoringEval</i> <i>at-</i> <i>tribute</i> ), 392	C_VERSION ( <i>mlpro.bf.streams.streams.clouds.StreamMLProClouds3D8C20</i> <i>attribute</i> ), 441
C_VAR_CYCLE ( <i>mlpro.gt.native.basics.GTDataStoring</i> <i>at-</i> <i>tribute</i> ), 415	C_VERSION ( <i>mlpro.bf.streams.streams.csv_file.StreamMLProCSV</i> <i>attribute</i> ), 435
C_VAR_CYCLE ( <i>mlpro.rl.models_train.RLDataStoring</i> <i>at-</i> <i>tribute</i> ), 392	C_VERSION ( <i>mlpro.bf.streams.streams.doublespiral2d.DoubleSpiral2D</i> <i>attribute</i> ), 437
C_VAR_DAY ( <i>mlpro.gt.native.basics.GTDataStoring</i> <i>at-</i> <i>tribute</i> ), 415	C_VERSION ( <i>mlpro.bf.streams.streams.point_outliers.StreamMLProPOutlie</i>

- attribute), 438
- C\_VERSION (mlpro.bf.streams.streams.rnd10d.StreamMLPro attribute), 436
- C\_VERSION (mlpro.bf.ui.sciui.framework.SciUIScenario attribute), 292
- C\_VERSION (mlpro.bf.ui.sciui.main.SciUI attribute), 285
- C\_VIEW\_2D (mlpro.bf.plot.PlotSettings attribute), 280
- C\_VIEW\_3D (mlpro.bf.plot.PlotSettings attribute), 280
- C\_VIEW\_ND (mlpro.bf.plot.PlotSettings attribute), 280
- C\_VISIBLE (mlpro.bf.ui.sciui.framework.SciUIScenario attribute), 292
- C\_WRAPPED\_PACKAGE (mlpro.wrappers.mujoco.MujocoHandler attribute), 516
- calc\_margin() (mlpro.rl.pool.envs.bglp.BGLP method), 483
- calc\_mass() (mlpro.rl.pool.envs.bglp.Belt method), 473
- calc\_mass() (mlpro.rl.pool.envs.bglp.VacuumPump method), 471
- calc\_mass\_flows() (mlpro.rl.pool.envs.bglp.BGLP method), 483
- calc\_power() (mlpro.rl.pool.envs.bglp.Belt method), 473
- calc\_power() (mlpro.rl.pool.envs.bglp.BGLP method), 483
- calc\_power() (mlpro.rl.pool.envs.bglp.VacuumPump method), 472
- calc\_reward() (mlpro.rl.pool.envs.bglp.BGLP method), 484
- calc\_state() (mlpro.rl.pool.envs.bglp.BGLP method), 483
- calculate\_metrics() (mlpro.sl.basics.SLAdaptiveFunction method), 371
- call\_mapping() (mlpro.gt.native.basics.GTPayoffMatrix method), 407
- CallbacksViewer (class in mlpro.wrappers.mujoco), 515
- cb\_mbutton\_pressed() (mlpro.bf.ui.sciui.framework.SciUISubplot2D method), 291
- cb\_mbutton\_released() (mlpro.bf.ui.sciui.framework.SciUISubplot2D method), 291
- cb\_mouse\_moved() (mlpro.bf.ui.sciui.framework.SciUISubplot2D method), 291
- cb\_popup\_menu() (mlpro.bf.ui.sciui.framework.SciUIFrame method), 289
- cb\_save\_plot() (mlpro.bf.ui.sciui.framework.SciUISubplotRoot method), 291
- change (mlpro.rl.pool.envs.bglp.Reservoir attribute), 474, 475
- checkin() (mlpro.bf.mt.Shared method), 298
- checkOut() (mlpro.bf.mt.Shared method), 298
- clear\_action\_path() (mlpro.rl.models\_agents.ActionPlanner method), 387
- clear\_buffer() (mlpro.bf.ml.basics.AWorkflow method), 362
- clear\_buffer() (mlpro.bf.ml.basics.Model method), 361
- clear\_buffer() (mlpro.rl.models\_agents.Agent method), 389
- clear\_buffer() (mlpro.rl.models\_agents.MultiAgent method), 390
- clear\_buffer() (mlpro.rl.models\_env\_ada.EnvModel method), 383
- clear\_results() (mlpro.bf.mt.Shared method), 298
- close() (mlpro.bf.ml.basics.TrainingResults method), 364
- close() (mlpro.rl.models\_train.RLTrainingResults method), 396
- close() (mlpro.wrappers.mujoco.BaseViewer method), 515
- close() (mlpro.wrappers.mujoco.OffRenderViewer method), 515
- close() (mlpro.wrappers.mujoco.RenderViewer method), 515
- collect\_substates() (mlpro.rl.pool.envs.bglp.BGLP method), 482
- compute\_action() (mlpro.gt.dynamicgames.stackelberg.GTMultiPlayer\_SG method), 402
- compute\_action() (mlpro.gt.dynamicgames.stackelberg.GTPlayer\_SG method), 401
- compute\_action() (mlpro.rl.models\_agents.ActionPlanner method), 386
- compute\_action() (mlpro.rl.models\_agents.Agent method), 389
- compute\_action() (mlpro.rl.models\_agents.MultiAgent method), 390
- compute\_action() (mlpro.rl.models\_agents.Policy method), 385
- compute\_broken() (mlpro.bf.systems.basics.FctBroken method), 342
- compute\_broken() (mlpro.bf.systems.basics.MultiSystem method), 354
- compute\_broken() (mlpro.bf.systems.basics.System method), 352
- compute\_potential() (mlpro.gt.dynamicgames.potential.PGameBoard method), 401

<code>compute_reward()</code> ( <i>mlpro.rl.models_env.EnvBase method</i> ), 378	<code>pro.bf.ui.sciui.framework.SciUISubplot3D method), 291</code>
<code>compute_reward()</code> ( <i>mlpro.rl.models_env.FctReward method</i> ), 376	<code>create_subplot()</code> ( <i>mlpro.bf.ui.sciui.framework.SciUISubplotRoot method</i> ), 290
<code>compute_reward_001()</code> ( <i>mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot method</i> ), 488	<code>cur_action</code> ( <i>mlpro.rl.pool.envs.bglp.Actuator attribute</i> ), 469, 470
<code>compute_reward_002()</code> ( <i>mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot method</i> ), 488	<code>cur_mass_transport</code> ( <i>mlpro.rl.pool.envs.bglp.Actuator attribute</i> ), 469, 470
<code>compute_reward_003()</code> ( <i>mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot method</i> ), 489	<code>cur_power</code> ( <i>mlpro.rl.pool.envs.bglp.Actuator attribute</i> ), 469, 470
<code>compute_reward_004()</code> ( <i>mlpro.rl.pool.envs.doublependulum.DoublePendulumRoot method</i> ), 489	<code>cur_speed</code> ( <i>mlpro.rl.pool.envs.bglp.Actuator attribute</i> ), 470
<code>compute_strategy()</code> ( <i>mlpro.gt.native.basics.GTCoalition method</i> ), 413	<code>cycle_limit</code> ( <i>mlpro.rl.pool.envs.bglp.BGLP attribute</i> ), 480
<code>compute_strategy()</code> ( <i>mlpro.gt.native.basics.GTCompetition method</i> ), 414	<b>D</b>
<code>compute_strategy()</code> ( <i>mlpro.gt.native.basics.GTPlayer method</i> ), 410	<code>DataObject</code> ( <i>class in mlpro.bf.math.basics</i> ), 311
<code>compute_strategy()</code> ( <i>mlpro.gt.native.basics.GTSolver method</i> ), 409	<code>DataPlotting</code> ( <i>class in mlpro.bf.plot</i> ), 283
<code>compute_success()</code> ( <i>mlpro.bf.systems.basics.FctSuccess method</i> ), 342	<code>deactivate()</code> ( <i>mlpro.rl.pool.envs.bglp.Belt method</i> ), 474
<code>compute_success()</code> ( <i>mlpro.bf.systems.basics.MultiSystem method</i> ), 354	<code>deactivate()</code> ( <i>mlpro.rl.pool.envs.bglp.VacuumPump method</i> ), 472
<code>compute_success()</code> ( <i>mlpro.bf.systems.basics.System method</i> ), 351	<code>demand</code> ( <i>mlpro.rl.pool.envs.bglp.BGLP attribute</i> ), 481
<code>con_act_to_res</code> ( <i>mlpro.rl.pool.envs.bglp.BGLP attribute</i> ), 478, 480	<code>demand_t</code> ( <i>mlpro.rl.pool.envs.bglp.BGLP attribute</i> ), 479, 481
<code>connect_data_logger()</code> ( <i>mlpro.gt.native.basics.GTGame method</i> ), 417	<code>DemoScenario</code> ( <i>class in mlpro.bf.systems.basics</i> ), 354
<code>connect_data_logger()</code> ( <i>mlpro.rl.models_train.RLScenario method</i> ), 395	<code>denormalize()</code> ( <i>mlpro.bf.math.normalizers.Normalizer method</i> ), 314
<code>connect_event()</code> ( <i>mlpro.bf.ui.sciui.framework.SciUICursor method</i> ), 288	<code>DerivativeFunction</code> ( <i>class in mlpro.bf.streams.tasks.deriver</i> ), 443
<code>Controller</code> ( <i>class in mlpro.bf.systems.basics</i> ), 342	<code>Deriver</code> ( <i>class in mlpro.bf.streams.tasks.deriver</i> ), 442
<code>copy()</code> ( <i>mlpro.bf.math.basics.Dimension method</i> ), 310	<code>determine_frame_size()</code> ( <i>mlpro.bf.ui.sciui.framework.SciUIFrame method</i> ), 289
<code>copy()</code> ( <i>mlpro.bf.math.basics.Element method</i> ), 311	<code>determine_frame_size()</code> ( <i>mlpro.bf.ui.sciui.framework.SciUISubplotRoot method</i> ), 290
<code>copy()</code> ( <i>mlpro.bf.math.basics.Set method</i> ), 311	<code>Dimension</code> ( <i>class in mlpro.bf.math.basics</i> ), 309
<code>copy()</code> ( <i>mlpro.bf.streams.models.Instance method</i> ), 319	<code>distance()</code> ( <i>mlpro.bf.math.basics.ESpace method</i> ), 312
<code>copy()</code> ( <i>mlpro.bf.systems.basics.State method</i> ), 340	<code>distance()</code> ( <i>mlpro.bf.math.basics.MSpace method</i> ), 312
<code>create_subplot()</code> ( <i>mlpro.bf.ui.sciui.framework.SciUISubplot2D method</i> ), 291	<code>DoublePendulumOA4</code> ( <i>class in mlpro.rl.pool.envs.doublependulum</i> ), 493
<code>create_subplot()</code> ( <i>mlpro.bf.ui.sciui.framework.SciUISubplot3D method</i> ), 291	<code>DoublePendulumOA7</code> ( <i>class in mlpro.rl.pool.envs.doublependulum</i> ), 494
	<code>DoublePendulumRoot</code> ( <i>class in mlpro.rl.pool.envs.doublependulum</i> ), 486
	<code>DoublePendulumS4</code> ( <i>class in mlpro.rl.pool.envs.doublependulum</i> ), 489
	<code>DoublePendulumS7</code> ( <i>class in mlpro.rl.pool.envs.doublependulum</i> ), 491
	<code>DoublePendulumSystemRoot</code> ( <i>class in mlpro.bf.systems.pool.doublependulum</i> ), 453

DoublePendulumSystemS4 (class in *mlpro.bf.systems.pool.doublependulum*), 456  
 DoublePendulumSystemS7 (class in *mlpro.bf.systems.pool.doublependulum*), 458  
 DoubleSpiral2D (class in *mlpro.bf.streams.streams.doublespiral2d*), 437

## E

Element (class in *mlpro.bf.math.basics*), 311  
 ElementList (class in *mlpro.bf.math.basics*), 311  
 enter() (*mlpro.bf.ui.sciui.framework.SciUITooltip* method), 288  
 EnvBase (class in *mlpro.rl.models\_env*), 377  
 Environment (class in *mlpro.rl.models\_env*), 379  
 EnvModel (class in *mlpro.rl.models\_env\_ada*), 382  
 Error, 285  
 ESpace (class in *mlpro.bf.math.basics*), 312  
 Event (class in *mlpro.bf.events*), 293  
 event\_forwarder() (*mlpro.bf.mt.Workflow* method), 303  
 EventManager (class in *mlpro.bf.events*), 294  
 execute() (*mlpro.rl.pool.actionplanner.mpc.MPC* method), 468

## F

FctBroken (class in *mlpro.bf.systems.basics*), 342  
 FctReward (class in *mlpro.rl.models\_env*), 376  
 FctSTrans (class in *mlpro.bf.systems.basics*), 341  
 FctSuccess (class in *mlpro.bf.systems.basics*), 341  
 Feature (class in *mlpro.bf.streams.models*), 319  
 finish\_lap() (*mlpro.bf.various.Timer* method), 275  
 FNN (class in *mlpro.sl.fnn*), 372  
 force\_fg() (*mlpro.bf.plot.Plottable* method), 281  
 forward() (*mlpro.sl.fnn.FNN* method), 372  
 forward() (*mlpro.sl.pool.afct.fnn.pytorch.mlp.PyTorchMLP* method), 464  
 free() (*mlpro.wrappers.mujoco.OffRenderViewer* method), 515  
 free() (*mlpro.wrappers.mujoco.RenderViewer* method), 515  
 Function (class in *mlpro.bf.math.basics*), 312

## G

Game (class in *mlpro.gt.dynamicgames.basics*), 399  
 GameBoard (class in *mlpro.gt.dynamicgames.basics*), 398  
 get\_acceleration() (*mlpro.bf.math.geometry.Point* method), 316  
 get\_accuracy() (*mlpro.bf.ml.basics.AWorkflow* method), 362  
 get\_accuracy() (*mlpro.bf.ml.basics.Model* method), 361  
 get\_accuracy() (*mlpro.rl.models\_env\_ada.EnvModel* method), 383

get\_accuracy() (*mlpro.sl.basics.SLAdaptiveFunction* method), 371  
 get\_action() (*mlpro.bf.systems.basics.SystemShared* method), 345  
 get\_action\_reward() (*mlpro.rl.models\_env.Reward* method), 376  
 get\_action\_space() (*mlpro.bf.systems.basics.System* method), 349  
 get\_action\_space() (*mlpro.rl.models\_agents.Agent* method), 388  
 get\_action\_space() (*mlpro.rl.models\_agents.MultiAgent* method), 390  
 get\_action\_space() (*mlpro.rl.models\_agents.Policy* method), 385  
 get\_actions() (*mlpro.bf.systems.basics.SystemShared* method), 345  
 get\_actuator() (*mlpro.bf.systems.basics.Controller* method), 344  
 get\_actuators() (*mlpro.bf.systems.basics.Controller* method), 344  
 get\_adapted() (*mlpro.bf.ml.basics.AWorkflow* method), 362  
 get\_adapted() (*mlpro.bf.ml.basics.Model* method), 360  
 get\_adapted() (*mlpro.rl.models\_env\_ada.EnvModel* method), 383  
 get\_agent() (*mlpro.rl.models\_agents.MultiAgent* method), 390  
 get\_agent() (*mlpro.rl.models\_train.RLScenario* method), 395  
 get\_agent\_ids() (*mlpro.bf.systems.basics.Action* method), 341  
 get\_agent\_reward() (*mlpro.rl.models\_env.Reward* method), 376  
 get\_agents() (*mlpro.rl.models\_agents.MultiAgent* method), 390  
 get\_all() (*mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.PrioritizedBuffer* method), 499  
 get\_all\_states() (*mlpro.rl.pool.envs.gridworld.GridWorld* method), 496  
 get\_base\_set() (*mlpro.bf.math.basics.Dimension* method), 309  
 get\_bibtex() (*mlpro.bf.various.ScientificObject* method), 277  
 get\_boundaries() (*mlpro.bf.math.basics.Dimension* method), 310  
 get\_boundaries() (*mlpro.bf.streams.tasks.windows.Window* method), 446  
 get\_broken() (*mlpro.bf.systems.basics.State* method), 340  
 get\_broken() (*mlpro.bf.systems.basics.System* method), 352



[get\\_buffered\\_data\(\)](#) (*mlpro.bf.streams.tasks.windows.Window* method), 445  
[get\\_coalition\(\)](#) (*mlpro.gt.native.basics.GTCompetition* method), 414  
[get\\_coalition\\_strategy\(\)](#) (*mlpro.gt.native.basics.GTCoalition* method), 412  
[get\\_coalitions\(\)](#) (*mlpro.gt.native.basics.GTCompetition* method), 413  
[get\\_coalitions\\_ids\(\)](#) (*mlpro.gt.native.basics.GTCompetition* method), 414  
[get\\_current\\_metrics\(\)](#) (*mlpro.sl.basics.SLAdaptiveFunction* method), 371  
[get\\_cycle\\_id\(\)](#) (*mlpro.bf.ops.ScenarioBase* method), 306  
[get\\_cycle\\_limit\(\)](#) (*mlpro.rl.models\_env.EnvBase* method), 378  
[get\\_cycle\\_limit\(\)](#) (*mlpro.rl.models\_env.Environment* method), 380  
[get\\_cycle\\_limit\(\)](#) (*mlpro.rl.models\_env\_ada.EnvModel* method), 383  
[get\\_data\(\)](#) (*mlpro.bf.events.Event* method), 293  
[get\\_data\(\)](#) (*mlpro.bf.math.basics.DataObject* method), 311  
[get\\_description\(\)](#) (*mlpro.bf.math.basics.Dimension* method), 310  
[get\\_dim\(\)](#) (*mlpro.bf.math.basics.Set* method), 310  
[get\\_dim\\_by\\_name\(\)](#) (*mlpro.bf.math.basics.Set* method), 310  
[get\\_dim\\_ids\(\)](#) (*mlpro.bf.math.basics.Element* method), 311  
[get\\_dim\\_ids\(\)](#) (*mlpro.bf.math.basics.Set* method), 310  
[get\\_dims\(\)](#) (*mlpro.bf.math.basics.Set* method), 310  
[get\\_elem\(\)](#) (*mlpro.bf.math.basics.ElementList* method), 312  
[get\\_elem\\_ids\(\)](#) (*mlpro.bf.math.basics.ElementList* method), 312  
[get\\_env\(\)](#) (*mlpro.rl.models\_train.RLScenario* method), 395  
[get\\_fct\\_broken\(\)](#) (*mlpro.bf.systems.basics.System* method), 349  
[get\\_fct\\_strans\(\)](#) (*mlpro.bf.systems.basics.System* method), 349  
[get\\_fct\\_success\(\)](#) (*mlpro.bf.systems.basics.System* method), 349  
[get\\_feature\\_data\(\)](#) (*mlpro.bf.streams.models.Instance* method), 319  
[get\\_feature\\_data\(\)](#) (*mlpro.bf.systems.basics.State* method), 340  
[get\\_feature\\_space\(\)](#) (*mlpro.bf.streams.models.Stream* method), 322  
[get\\_filename\(\)](#) (*mlpro.bf.ui.sciui.framework.SciUISubplotSaveDLG* method), 290  
[get\\_filename\(\)](#) (*mlpro.bf.various.Persistent* method), 273  
[get\\_filename\\_stub\(\)](#) (*mlpro.bf.various.Persistent* method), 273  
[get\\_functions\(\)](#) (*mlpro.rl.models\_env.EnvBase* method), 378  
[get\\_homogeneous\(\)](#) (*mlpro.rl.pool.envs.robotinhtm.RobotArm3D* method), 497  
[get\\_homogeneous\\_eef\(\)](#) (*mlpro.rl.pool.envs.robotinhtm.RobotArm3D* method), 497  
[get\\_hyperparam\(\)](#) (*mlpro.bf.ml.basics.Model* method), 359  
[get\\_hyperparam\(\)](#) (*mlpro.gt.native.basics.GTSolver* method), 409  
[get\\_id\(\)](#) (*mlpro.bf.math.basics.Dimension* method), 309  
[get\\_id\(\)](#) (*mlpro.bf.streams.models.Instance* method), 319  
[get\\_id\(\)](#) (*mlpro.bf.various.Id* method), 272  
[get\\_init\\_qpos\\_space\(\)](#) (*mlpro.wrappers.mujoco.MujocoHandler* method), 516  
[get\\_init\\_qvel\\_space\(\)](#) (*mlpro.wrappers.mujoco.MujocoHandler* method), 516  
[get\\_initial\(\)](#) (*mlpro.bf.systems.basics.State* method), 340  
[get\\_input\\_space\(\)](#) (*mlpro.sl.basics.SLAdaptiveFunction* method), 370  
[get\\_instances\(\)](#) (*mlpro.bf.streams.models.StreamShared* method), 320  
[get\\_internal\\_counter\(\)](#) (*mlpro.sl.pool.afct.pytorch.PyTorchBuffer* method), 465  
[get\\_joint\(\)](#) (*mlpro.rl.pool.envs.robotinhtm.RobotArm3D* method), 497  
[get\\_kwargs\(\)](#) (*mlpro.bf.math.basics.Dimension* method), 310  
[get\\_kwargs\(\)](#) (*mlpro.bf.streams.models.Instance* method), 319  
[get\\_label\\_data\(\)](#) (*mlpro.bf.streams.models.Instance* method), 319  
[get\\_label\\_space\(\)](#) (*mlpro.bf.streams.models.Stream*

- method), 322
- get\_lap\_id() (*mlpro.bf.various.Timer* method), 275
- get\_lap\_time() (*mlpro.bf.various.Timer* method), 275
- get\_last\_reward() (*mlpro.rl.models\_env.EnvBase* method), 378
- get\_latency() (*mlpro.bf.ops.ScenarioBase* method), 305
- get\_latency() (*mlpro.bf.streams.models.StreamScenario* method), 329
- get\_latency() (*mlpro.bf.systems.basics.DemoScenario* method), 355
- get\_latency() (*mlpro.bf.systems.basics.System* method), 348
- get\_latency() (*mlpro.gt.native.basics.GTGame* method), 417
- get\_latency() (*mlpro.rl.models\_train.RLScenario* method), 395
- get\_latency() (*mlpro.wrappers.mujoco.MujocoHandler* method), 518
- get\_latest() (*mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.PriorityBuffer* method), 499
- get\_log\_level() (*mlpro.bf.various.Log* method), 273
- get\_logging\_data() (*mlpro.sl.basics.SLAdaptiveFunction* method), 370
- get\_logging\_set() (*mlpro.sl.basics.SLAdaptiveFunction* method), 370
- get\_mean() (*mlpro.bf.streams.tasks.windows.Window* method), 446
- get\_meta\_data() (*mlpro.bf.math.basics.DataObject* method), 311
- get\_metric\_space() (*mlpro.sl.basics.SLAdaptiveFunction* method), 371
- get\_metrics() (*mlpro.sl.basics.SLAdaptiveFunction* method), 371
- get\_mode() (*mlpro.bf.ops.Mode* method), 304
- get\_model() (*mlpro.bf.ml.basics.Scenario* method), 363
- get\_model() (*mlpro.sl.basics.SLAdaptiveFunction* method), 370
- get\_name() (*mlpro.bf.streams.models.Stream* method), 321
- get\_name() (*mlpro.bf.ui.sciui.framework.SciUIComponent* method), 288
- get\_name() (*mlpro.bf.various.Log* method), 272
- get\_name() (*mlpro.bf.various.PersonalisedStamp* method), 277
- get\_name\_latex() (*mlpro.bf.math.basics.Dimension* method), 310
- get\_name\_long() (*mlpro.bf.math.basics.Dimension* method), 310
- get\_name\_short() (*mlpro.bf.math.basics.Dimension* method), 309
- get\_num\_dim() (*mlpro.bf.math.basics.Set* method), 310
- get\_num\_instances() (*mlpro.bf.streams.models.Sampler* method), 320
- get\_num\_instances() (*mlpro.bf.streams.models.Stream* method), 322
- get\_num\_joint() (*mlpro.rl.pool.envs.robotinhtm.RobotArm3D* method), 497
- get\_observation\_space() (*mlpro.rl.models\_agents.Agent* method), 388
- get\_observation\_space() (*mlpro.rl.models\_agents.MultiAgent* method), 390
- get\_observation\_space() (*mlpro.rl.models\_agents.Policy* method), 385
- get\_orientation() (*mlpro.rl.pool.envs.robotinhtm.RobotArm3D* method), 497
- get\_observation\_space() (*mlpro.sl.basics.SLAdaptiveFunction* method), 370
- get\_overall\_reward() (*mlpro.rl.models\_env.Reward* method), 376
- get\_payoff() (*mlpro.gt.native.basics.GTPayoffMatrix* method), 407
- get\_player() (*mlpro.gt.native.basics.GTCoalition* method), 412
- get\_player() (*mlpro.gt.native.basics.GTCompetition* method), 414
- get\_player\_ids() (*mlpro.gt.native.basics.GTStrategy* method), 405
- get\_players() (*mlpro.gt.native.basics.GTCoalition* method), 412
- get\_players() (*mlpro.gt.native.basics.GTCompetition* method), 414
- get\_players\_ids() (*mlpro.gt.native.basics.GTCoalition* method), 412
- get\_players\_ids() (*mlpro.gt.native.basics.GTCompetition* method), 414
- get\_plot\_settings() (*mlpro.bf.plot.Plottable* method), 281
- get\_plots() (*mlpro.bf.plot.DataPlotting* method), 284
- get\_position() (*mlpro.bf.math.geometry.Point* method), 316
- get\_predecessors() (*mlpro.bf.mt.Task* method), 301
- get\_raising\_object() (*mlpro.bf.events.Event* method), 293
- get\_range() (*mlpro.bf.mt.Range* method), 297
- get\_related\_set() (*mlpro.bf.math.basics.Element* method), 311
- get\_related\_set() (ml-

*pro.oa.streams.tasks.boundarydetectors.BoundaryDetector* *pro.gt.native.basics.GTPlayer* method), 410  
 method), 427  
*get\_result()* (*mlpro.bf.mt.Shared* method), 298  
*get\_results()* (*mlpro.bf.ml.basics.Training* method), 367  
*get\_results()* (*mlpro.bf.mt.Shared* method), 298  
*get\_reward\_type()* (*mlpro.rl.models\_env.EnvBase* method), 378  
*get\_scenario()* (*mlpro.bf.ml.basics.Training* method), 366  
*get\_score\_metric()* (*mlpro.sl.basics.SLAdaptiveFunction* method), 371  
*get\_sensor()* (*mlpro.bf.systems.basics.Controller* method), 343  
*get\_sensor\_value()* (*mlpro.bf.systems.basics.Controller* method), 343  
*get\_sensors()* (*mlpro.bf.systems.basics.Controller* method), 343  
*get\_so()* (*mlpro.bf.mt.Async* method), 299  
*get\_so()* (*mlpro.bf.systems.basics.System* method), 350  
*get\_solver()* (*mlpro.gt.native.basics.GTPlayer* method), 411  
*get\_sorted\_values()* (*mlpro.bf.systems.basics.Action* method), 341  
*get\_space()* (*mlpro.gt.native.basics.GTDataStoring* method), 415  
*get\_space()* (*mlpro.rl.models\_train.RLDataStoring* method), 392  
*get\_space()* (*mlpro.rl.models\_train.RLDataStoringEval* method), 393  
*get\_state()* (*mlpro.bf.systems.basics.System* method), 350  
*get\_state()* (*mlpro.bf.systems.basics.SystemShared* method), 346  
*get\_state\_space()* (*mlpro.bf.systems.basics.System* method), 349  
*get\_states()* (*mlpro.bf.systems.basics.MultiSystem* method), 353  
*get\_states()* (*mlpro.bf.systems.basics.SystemShared* method), 346  
*get\_status()* (*mlpro.rl.pool.envs.bglp.BGLP* method), 483  
*get\_std\_deviation()* (*mlpro.bf.streams.tasks.windows.Window* method), 446  
*get\_strategy\_space()* (*mlpro.gt.native.basics.GTCoalition* method), 412  
*get\_strategy\_space()* (*mlpro.gt.native.basics.GTCompetition* method), 415  
*get\_strategy\_space()* (*mlpro.gt.native.basics.GTPlayer* method), 410  
*get\_strategy\_space()* (*mlpro.gt.native.basics.GTSolver* method), 409  
*get\_stream()* (*mlpro.bf.streams.models.StreamProvider* method), 324  
*get\_stream()* (*mlpro.bf.streams.models.StreamScenario* method), 329  
*get\_stream\_list()* (*mlpro.bf.streams.models.StreamProvider* method), 323  
*get\_subsystem()* (*mlpro.bf.systems.basics.MultiSystem* method), 353  
*get\_subsystem\_ids()* (*mlpro.bf.systems.basics.MultiSystem* method), 353  
*get\_subsystems()* (*mlpro.bf.systems.basics.MultiSystem* method), 353  
*get\_success()* (*mlpro.bf.systems.basics.State* method), 340  
*get\_success()* (*mlpro.bf.systems.basics.System* method), 351  
*get\_symmetrical()* (*mlpro.bf.math.basics.Dimension* method), 310  
*get\_terminal()* (*mlpro.bf.systems.basics.State* method), 340  
*get\_tid()* (*mlpro.bf.mt.Task* method), 300  
*get\_time()* (*mlpro.bf.various.Timer* method), 275  
*get\_timeout()* (*mlpro.bf.systems.basics.State* method), 340  
*get\_training\_path()* (*mlpro.bf.ml.basics.Training* method), 367  
*get\_transformation\_matrix()* (*mlpro.rl.pool.envs.robotinhtl.RobotArm3D* method), 497  
*get\_tstamp()* (*mlpro.bf.various.TStamp* method), 275  
*get\_type()* (*mlpro.bf.physics.basics.TransferFunction* method), 333  
*get\_type()* (*mlpro.rl.models\_env.Reward* method), 376  
*get\_unit()* (*mlpro.bf.math.basics.Dimension* method), 310  
*get\_unit\_latex()* (*mlpro.bf.math.basics.Dimension* method), 310  
*get\_units()* (*mlpro.bf.physics.basics.TransferFunction* method), 333  
*get\_url()* (*mlpro.bf.streams.models.Stream* method), 321  
*get\_value()* (*mlpro.bf.math.basics.Element* method), 311  
*get\_value()* (*mlpro.bf.ml.basics.HyperParamDispatcher* method), 358  
*get\_values()* (*mlpro.bf.math.basics.Element* method), 311  
*get\_values()* (*mlpro.bf.ml.basics.HyperParamDispatcher*

- method*), 358
- `get_variables()` (*mlpro.gt.native.basics.GTDataStoring method*), 415
- `get_variables()` (*mlpro.rl.models\_train.RLDataStoring method*), 392
- `get_variables()` (*mlpro.rl.models\_train.RLDataStoringEval method*), 393
- `get_variance()` (*mlpro.bf.streams.tasks.windows.Window method*), 446
- `get_velocity()` (*mlpro.bf.math.geometry.Point method*), 316
- `get_visualization()` (*mlpro.bf.plot.Plottable method*), 281
- `get_weight()` (*mlpro.bf.systems.basics.ActionElement method*), 341
- `get_workflow()` (*mlpro.bf.streams.models.StreamScenario method*), 329
- `GridWorld` (*class in mlpro.rl.pool.envs.gridworld*), 495
- `GTCoalition` (*class in mlpro.gt.native.basics*), 411
- `GTCompetition` (*class in mlpro.gt.native.basics*), 413
- `GTDataStoring` (*class in mlpro.gt.native.basics*), 415
- `GTFunction` (*class in mlpro.gt.native.basics*), 405
- `GTGame` (*class in mlpro.gt.native.basics*), 415
- `GTMultiPlayer_SG` (*class in mlpro.gt.dynamicgames.stackelberg*), 402
- `GTPayoffMatrix` (*class in mlpro.gt.native.basics*), 406
- `GTPlayer` (*class in mlpro.gt.native.basics*), 409
- `GTPlayer_SG` (*class in mlpro.gt.dynamicgames.stackelberg*), 401
- `GTSolver` (*class in mlpro.gt.native.basics*), 408
- `GTStrategy` (*class in mlpro.gt.native.basics*), 405
- `GTTraining` (*class in mlpro.gt.dynamicgames.basics*), 399
- `GTTraining` (*class in mlpro.gt.native.basics*), 418
- `GTTrainingResults` (*class in mlpro.gt.dynamicgames.basics*), 399
- `GTTrainingResults` (*class in mlpro.gt.native.basics*), 417
- ## H
- `hidetip()` (*mlpro.bf.ui.sciui.framework.SciUITooltip method*), 288
- `Hopper` (*class in mlpro.rl.pool.envs.bglp*), 476
- `hops` (*mlpro.rl.pool.envs.bglp.BGLP attribute*), 478, 481
- `HyperParam` (*class in mlpro.bf.ml.basics*), 358
- `HyperParamDispatcher` (*class in mlpro.bf.ml.basics*), 358
- `HyperParamSpace` (*class in mlpro.bf.ml.basics*), 358
- `HyperParamTuner` (*class in mlpro.bf.ml.basics*), 365
- `HyperParamTuple` (*class in mlpro.bf.ml.basics*), 358
- ## I
- `Id` (*class in mlpro.bf.various*), 271
- `idx_a` (*mlpro.rl.pool.envs.bglp.Actuator attribute*), 468, 470
- `idx_b` (*mlpro.rl.pool.envs.bglp.Belt attribute*), 472, 473
- `idx_h` (*mlpro.rl.pool.envs.bglp.Hopper attribute*), 476
- `idx_r` (*mlpro.rl.pool.envs.bglp.Reservoir attribute*), 474
- `idx_s` (*mlpro.rl.pool.envs.bglp.Silo attribute*), 475, 476
- `idx_v` (*mlpro.rl.pool.envs.bglp.VacuumPump attribute*), 471
- `ImplementationError`, 285
- `init_component()` (*mlpro.bf.ui.sciui.framework.SciUIComponent method*), 288
- `init_component()` (*mlpro.bf.ui.sciui.framework.SciUIFrame method*), 289
- `init_component()` (*mlpro.bf.ui.sciui.framework.SciUIFrameParam method*), 292
- `init_component()` (*mlpro.bf.ui.sciui.framework.SciUISubplotRoot method*), 290
- `init_component()` (*mlpro.bf.ui.sciui.framework.SciUITabCTRL method*), 289
- `init_plot()` (*mlpro.bf.ml.basics.Scenario method*), 363
- `init_plot()` (*mlpro.bf.mt.Task method*), 301
- `init_plot()` (*mlpro.bf.mt.Workflow method*), 302
- `init_plot()` (*mlpro.bf.plot.Plottable method*), 281
- `init_plot()` (*mlpro.bf.streams.models.StreamScenario method*), 329
- `init_plot()` (*mlpro.bf.streams.models.StreamTask method*), 325
- `init_plot()` (*mlpro.bf.streams.models.StreamWorkflow method*), 327
- `init_plot()` (*mlpro.bf.systems.basics.System method*), 352
- `init_plot()` (*mlpro.rl.models\_agents.MultiAgent method*), 391
- `init_plot()` (*mlpro.rl.models\_train.RLScenario method*), 394
- `init_plot()` (*mlpro.rl.pool.envs.bglp.BGLP method*), 484
- `init_plot()` (*mlpro.rl.pool.envs.gridworld.GridWorld method*), 497
- `init_popup_menu()` (*mlpro.bf.ui.sciui.framework.SciUIFrame method*), 289
- `init_popup_menu()` (*mlpro.bf.ui.sciui.framework.SciUISubplotRoot method*), 290
- `input_preproc()` (*mlpro.sl.pool.afct.pytorch.PyTorchHelperFunctions*



- method), 466
- Instance (class in *mlpro.bf.streams.models*), 319
- is\_numeric() (*mlpro.bf.math.basics.Set* method), 310
- is\_rewarded() (*mlpro.rl.models\_env.Reward* method), 376
- is\_zerogsum() (*mlpro.gt.native.basics.GTGame* method), 417
- ## L
- Label (class in *mlpro.bf.streams.models*), 319
- leave() (*mlpro.bf.ui.sciui.framework.SciUITooltip* method), 288
- levels\_init (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 479, 481
- load() (*mlpro.bf.ml.basics.HyperParamTuner* class method), 365
- load() (*mlpro.bf.ml.basics.TrainingResults* class method), 364
- load() (*mlpro.bf.various.Persistent* class method), 274
- lock() (*mlpro.bf.mt.Shared* method), 298
- Log (class in *mlpro.bf.various*), 272
- log() (*mlpro.bf.various.Log* method), 273
- log\_results() (*mlpro.bf.ml.basics.TrainingResults* method), 364
- lr\_demand (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 479, 481
- lr\_margin (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 479, 481
- lr\_power (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 479, 481
- ## M
- m\_param (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 478, 480
- map() (*mlpro.bf.math.basics.Function* method), 312
- margin\_t (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 480, 481
- mass\_coeff (*mlpro.rl.pool.envs.bglp.Actuator* attribute), 469, 470
- MaxGreedyPolicy (class in *mlpro.gt.pool.native.solvers.greedypolicy*), 511
- MaxGreedyPolicy\_SD3P\_P1 (class in *mlpro.gt.pool.native.games.supplydemand\_3p*), 510
- MaxGreedyPolicy\_SD3P\_P2 (class in *mlpro.gt.pool.native.games.supplydemand\_3p*), 510
- maximize() (*mlpro.bf.ml.basics.HyperParamTuner* method), 365
- memorize\_row() (*mlpro.gt.native.basics.GTDataStoring* method), 415
- memorize\_row() (*mlpro.rl.models\_train.RLDataStoring* method), 392
- memorize\_row() (*mlpro.rl.models\_train.RLDataStoringEval* method), 393
- min() (*mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.MinSegmentTree* method), 500
- MinGreedyPolicy (class in *mlpro.gt.pool.native.solvers.greedypolicy*), 511
- MinGreedyPolicy\_Routing3P (class in *mlpro.gt.pool.native.games.routingproblems\_3p*), 507
- MinSegmentTree (class in *mlpro.rl.pool.sarsbuffer.PrioritizedBuffer*), 500
- MLP (class in *mlpro.sl.fun*), 373
- mlpro.bf.data module, 278
- mlpro.bf.events module, 293
- mlpro.bf.exceptions module, 285
- mlpro.bf.math.basics module, 308
- mlpro.bf.math.geometry module, 315
- mlpro.bf.math.normalizers module, 313
- mlpro.bf.ml.basics module, 357
- mlpro.bf.ml.systems module, 368
- mlpro.bf.mt module, 296
- mlpro.bf.ops module, 303
- mlpro.bf.physics.basics module, 331
- mlpro.bf.physics.unitconverter module, 335
- mlpro.bf.plot module, 279
- mlpro.bf.streams.models module, 318
- mlpro.bf.streams.samplers.min\_wise module, 447
- mlpro.bf.streams.samplers.random module, 448
- mlpro.bf.streams.samplers.reservoir\_sampling module, 449
- mlpro.bf.streams.samplers.weighted\_random module, 450
- mlpro.bf.streams.streams.clouds module, 28, 439
- mlpro.bf.streams.streams.csv\_file module, 27, 434

mlpro.bf.streams.streams.doublespiral2d  
module, 437

mlpro.bf.streams.streams.point\_outliers  
module, 28, 438

mlpro.bf.streams.streams.rnd10d  
module, 28, 436

mlpro.bf.streams.tasks.deriver  
module, 442

mlpro.bf.streams.tasks.rearranger  
module, 444

mlpro.bf.streams.tasks.windows  
module, 445

mlpro.bf.systems.basics  
module, 339

mlpro.bf.systems.pool.doublependulum  
module, 452

mlpro.bf.ui.sciui.framework  
module, 287

mlpro.bf.ui.sciui.main  
module, 285

mlpro.bf.various  
module, 271

mlpro.gt.dynamicgames.basics  
module, 398

mlpro.gt.dynamicgames.potential  
module, 400

mlpro.gt.dynamicgames.stackelberg  
module, 401

mlpro.gt.native.basics  
module, 404

mlpro.gt.pool.boards.bglp  
module, 501

mlpro.gt.pool.native.games.prisonersdilemma\_2p  
module, 501

mlpro.gt.pool.native.games.prisonersdilemma\_3p  
module, 502

mlpro.gt.pool.native.games.rockpaperscissors  
module, 504

mlpro.gt.pool.native.games.routingproblems\_3p  
module, 505

mlpro.gt.pool.native.games.supplydemand\_3p  
module, 508

mlpro.gt.pool.native.solvers.greedypolicy  
module, 511

mlpro.gt.pool.native.solvers.randomsolver  
module, 512

mlpro.oa.streams.basics  
module, 420, 431

mlpro.oa.streams.tasks.anomalydetectors  
module, 431

mlpro.oa.streams.tasks.boundarydetectors  
module, 426

mlpro.oa.streams.tasks.clusteranalyzers  
module, 430

mlpro.oa.streams.tasks.normalizers  
module, 427

mlpro.rl.models\_agents  
module, 384

mlpro.rl.models\_env  
module, 375

mlpro.rl.models\_env\_ada  
module, 380

mlpro.rl.models\_train  
module, 391

mlpro.rl.pool.actionplanner.mpc  
module, 76, 467

mlpro.rl.pool.envs.bglp  
module, 60, 468

mlpro.rl.pool.envs.doublependulum  
module, 68, 485

mlpro.rl.pool.envs.gridworld  
module, 65, 495

mlpro.rl.pool.envs.robotinhtm  
module, 67, 497

mlpro.rl.pool.policies.randomgenerator  
module, 76

mlpro.rl.pool.sarsbuffer.PrioritizedBuffer  
module, 499

mlpro.rl.pool.sarsbuffer.RandomSARSBBuffer  
module, 500

mlpro.sl.basics  
module, 369

mlpro.sl.fnn  
module, 372

mlpro.sl.pool.afct.fnn.pytorch.mlp  
module, 461

mlpro.sl.pool.afct.pytorch  
module, 465

mlpro.wrappers.mujoco  
module, 514

Mode (*class in mlpro.bf.ops*), 304

Model (*class in mlpro.bf.ml.basics*), 358

module

mlpro.bf.data, 278

mlpro.bf.events, 293

mlpro.bf.exceptions, 285

mlpro.bf.math.basics, 308

mlpro.bf.math.geometry, 315

mlpro.bf.math.normalizers, 313

mlpro.bf.ml.basics, 357

mlpro.bf.ml.systems, 368

mlpro.bf.mt, 296

mlpro.bf.ops, 303

mlpro.bf.physics.basics, 331

mlpro.bf.physics.unitconverter, 335

mlpro.bf.plot, 279

mlpro.bf.streams.models, 318

mlpro.bf.streams.samplers.min\_wise, 447

mlpro.bf.streams.samplers.random, 448  
 mlpro.bf.streams.samplers.reservoir\_sampling, 449  
 mlpro.bf.streams.samplers.weighted\_random, 450  
 mlpro.bf.streams.streams.clouds, 28, 439  
 mlpro.bf.streams.streams.csv\_file, 27, 434  
 mlpro.bf.streams.streams.doublespiral2d, 437  
 mlpro.bf.streams.streams.point\_outliers, 28, 438  
 mlpro.bf.streams.streams.rnd10d, 28, 436  
 mlpro.bf.streams.tasks.deriver, 442  
 mlpro.bf.streams.tasks.rearranger, 444  
 mlpro.bf.streams.tasks.windows, 445  
 mlpro.bf.systems.basics, 339  
 mlpro.bf.systems.pool.doublependulum, 452  
 mlpro.bf.ui.sciui.framework, 287  
 mlpro.bf.ui.sciui.main, 285  
 mlpro.bf.various, 271  
 mlpro.gt.dynamicgames.basics, 398  
 mlpro.gt.dynamicgames.potential, 400  
 mlpro.gt.dynamicgames.stackelberg, 401  
 mlpro.gt.native.basics, 404  
 mlpro.gt.pool.boards.bglp, 501  
 mlpro.gt.pool.native.games.prisonersdilemma, 501  
 mlpro.gt.pool.native.games.prisonersdilemma\_3p, 502  
 mlpro.gt.pool.native.games.rockpaperscissors, 504  
 mlpro.gt.pool.native.games.routingproblems\_3p, 505  
 mlpro.gt.pool.native.games.supplydemand\_3p, 508  
 mlpro.gt.pool.native.solvers.greedypolicy, 511  
 mlpro.gt.pool.native.solvers.randomsolver, 512  
 mlpro.oa.streams.basics, 420, 431  
 mlpro.oa.streams.tasks.anomalydetectors, 431  
 mlpro.oa.streams.tasks.boundarydetectors, 426  
 mlpro.oa.streams.tasks.clusteranalyzers, 430  
 mlpro.oa.streams.tasks.normalizers, 427  
 mlpro.rl.models\_agents, 384  
 mlpro.rl.models\_env, 375  
 mlpro.rl.models\_env\_ada, 380  
 mlpro.rl.models\_train, 391  
 mlpro.rl.pool.actionplanner.mpc, 76, 467  
 mlpro.rl.pool.envs.bglp, 60, 468  
 mlpro.rl.pool.envs.doublependulum, 68, 485  
 mlpro.rl.pool.envs.gridworld, 65, 495  
 mlpro.rl.pool.envs.robotinhtn, 67, 497  
 mlpro.rl.pool.policies.randomgenerator, 76  
 mlpro.rl.pool.sarsbuffer.PrioritizedBuffer, 499  
 mlpro.rl.pool.sarsbuffer.RandomSARSBUFFER, 500  
 mlpro.sl.basics, 369  
 mlpro.sl.fnn, 372  
 mlpro.sl.pool.afct.fnn.pytorch.mlp, 461  
 mlpro.sl.pool.afct.pytorch, 465  
 mlpro.wrappers.mujoco, 514  
 moving\_mean() (*mlpro.bf.plot.DataPlotting* method), 284  
 MPC (*class in mlpro.rl.pool.actionplanner.mpc*), 467  
 MSpace (*class in mlpro.bf.math.basics*), 312  
 MujocoHandler (*class in mlpro.wrappers.mujoco*), 515  
 MultiAgent (*class in mlpro.rl.models\_agents*), 389  
 MultiPlayer (*class in mlpro.gt.dynamicgames.basics*), 399  
 MultiSystem (*class in mlpro.bf.systems.basics*), 352  

## N

 name (*mlpro.rl.pool.envs.bglp.Belt* attribute), 472, 473  
 name (*mlpro.rl.pool.envs.bglp.Hopper* attribute), 476  
 name (*mlpro.rl.pool.envs.bglp.Silo* attribute), 475, 476  
 name (*mlpro.rl.pool.envs.bglp.VacuumPump* attribute), 471  
 normalize() (*mlpro.bf.math.normalizers.Normalizer* method), 314  
 Normalizer (*class in mlpro.bf.math.normalizers*), 313  
 NormalizerMinMax (*class in mlpro.bf.math.normalizers*), 314  
 NormalizerMinMax (*class in mlpro.oa.streams.tasks.normalizers*), 427  
 NormalizerZTrans (*class in mlpro.bf.math.normalizers*), 314  
 NormalizerZTransform (*class in mlpro.oa.streams.tasks.normalizers*), 429  

## O

 OAScenario (*class in mlpro.oa.streams.basics*), 423, 433  
 OAShared (*class in mlpro.oa.streams.basics*), 420, 431  
 OATask (*class in mlpro.oa.streams.basics*), 420, 431  
 OATraining (*class in mlpro.oa.streams.basics*), 424, 434  
 OATrainingResults (*class in mlpro.oa.streams.basics*), 423, 434  
 OAWorkflow (*class in mlpro.oa.streams.basics*), 422, 432  
 OffRenderViewer (*class in mlpro.wrappers.mujoco*), 515  
 omit\_instance() (*mlpro.bf.streams.models.Sampler* method), 320

- onbuttonpressed() (mlpro.bf.ui.sciui.framework.SciUICursor method), 288
- onbuttonreleased() (mlpro.bf.ui.sciui.framework.SciUICursor method), 288
- onmove() (mlpro.bf.ui.sciui.framework.SciUICursor method), 288
- operate() (mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.Segment method), 500
- operation (mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.Segment attribute), 500
- output\_postproc() (mlpro.sl.pool.afct.pytorch.PyTorchHelperFunctions method), 466
- output\_preproc() (mlpro.sl.pool.afct.pytorch.PyTorchHelperFunctions method), 466
- overflow (mlpro.rl.pool.envs.bglp.BGLP attribute), 479, 481
- overflow\_t (mlpro.rl.pool.envs.bglp.BGLP attribute), 479, 481
- ## P
- ParamError, 285
- PayoffFunction\_PD2P (class in mlpro.gt.pool.native.games.prisonersdilemma\_2p), 501
- PayoffFunction\_PD3P (class in mlpro.gt.pool.native.games.prisonersdilemma\_3p), 503
- PayoffFunction\_Routing3P (class in mlpro.gt.pool.native.games.routingproblems\_3p), 506
- PayoffFunction\_RSP (class in mlpro.gt.pool.native.games.rockpaperscissors), 504
- PayoffFunction\_SD3P (class in mlpro.gt.pool.native.games.supplydemand\_3p), 509
- PayoffMatrix\_Routing3P (class in mlpro.gt.pool.native.games.routingproblems\_3p), 506
- PayoffMatrix\_SD3P (class in mlpro.gt.pool.native.games.supplydemand\_3p), 509
- Persistent (class in mlpro.bf.various), 273
- PersonalisedStamp (class in mlpro.bf.various), 277
- PGameBoard (class in mlpro.gt.dynamicgames.potential), 400
- Player (class in mlpro.gt.dynamicgames.basics), 399
- plot() (mlpro.bf.physics.basics.TransferFunction method), 334
- plots\_type\_cy() (mlpro.bf.plot.DataPlotting method), 284
- plots\_type\_ep() (mlpro.bf.plot.DataPlotting method), 284
- plots\_type\_ep\_mean() (mlpro.bf.plot.DataPlotting method), 284
- PlotSettings (class in mlpro.bf.plot), 279
- Plottable (class in mlpro.bf.plot), 280
- Point (class in mlpro.bf.math.geometry), 316
- Policy (class in mlpro.rl.models\_agents), 384
- power (mlpro.rl.pool.envs.bglp.BGLP attribute), 479, 481
- power\_coeff (mlpro.rl.pool.envs.bglp.Actuator attribute), 469, 470
- power\_max (mlpro.rl.pool.envs.bglp.Actuator attribute), 468, 470
- power\_min (mlpro.rl.pool.envs.bglp.Actuator attribute), 469, 470
- power\_t (mlpro.rl.pool.envs.bglp.BGLP attribute), 480, 481
- PrioritizedBuffer (class in mlpro.rl.pool.sarsbuffer.PrioritizedBuffer), 499
- PrioritizedBufferElement (class in mlpro.rl.pool.sarsbuffer.PrioritizedBuffer), 499
- PrisonersDilemma2PGame (class in mlpro.gt.pool.native.games.prisonersdilemma\_2p), 502
- PrisonersDilemma3PGame (class in mlpro.gt.pool.native.games.prisonersdilemma\_3p), 503
- process\_action() (mlpro.bf.systems.basics.System method), 350
- prod\_reached (mlpro.rl.pool.envs.bglp.BGLP attribute), 480
- prod\_scenario (mlpro.rl.pool.envs.bglp.BGLP attribute), 480, 481
- prod\_target (mlpro.rl.pool.envs.bglp.BGLP attribute), 480, 481
- PyTorchBuffer (class in mlpro.sl.pool.afct.pytorch), 465
- PyTorchHelperFunctions (class in mlpro.sl.pool.afct.pytorch), 466
- PyTorchIOElement (class in mlpro.sl.pool.afct.pytorch), 465
- PyTorchMLP (class in mlpro.sl.pool.afct.fnn.pytorch.mlp), 461
- ## R
- RandomSARSBuffer (class in mlpro.rl.pool.sarsbuffer.RandomSARSBuffer), 500
- RandomSolver (class in mlpro.gt.pool.native.solvers.randomsolver), 500



- 512
- Range (class in *mlpro.bf.mt*), 297
- Rearranger (class in *mlpro.bf.streams.tasks.rearranger*), 444
- refresh() (*mlpro.bf.ui.sciui.framework.SciUIComponent* method), 288
- refresh() (*mlpro.bf.ui.sciui.framework.SciUIFrame* method), 289
- refresh() (*mlpro.bf.ui.sciui.framework.SciUISubplotRoot* method), 290
- refresh() (*mlpro.bf.ui.sciui.framework.SciUITabCTRL* method), 289
- refresh\_custom() (*mlpro.bf.ui.sciui.framework.SciUISubplotRoot* method), 291
- refresh\_plot() (*mlpro.bf.plot.Plottable* method), 282
- reg\_a (*mlpro.rl.pool.envs.bglp.Actuator* attribute), 468, 470
- reg\_b (*mlpro.rl.pool.envs.bglp.Belt* attribute), 472, 473
- reg\_h (*mlpro.rl.pool.envs.bglp.Hopper* attribute), 476
- reg\_r (*mlpro.rl.pool.envs.bglp.Reservoir* attribute), 474
- reg\_s (*mlpro.rl.pool.envs.bglp.Silo* attribute), 475
- reg\_v (*mlpro.rl.pool.envs.bglp.VacuumPump* attribute), 471
- register\_event\_handler() (*mlpro.bf.events.EventManager* method), 294
- register\_scenario() (*mlpro.bf.ui.sciui.main.SciUI* method), 286
- register\_system() (*mlpro.bf.systems.basics.SystemShared* method), 346
- remove\_event\_handler() (*mlpro.bf.events.EventManager* method), 294
- remove\_plot() (*mlpro.bf.plot.Plottable* method), 283
- render() (*mlpro.wrappers.mujoco.MujocoHandler* method), 518
- render() (*mlpro.wrappers.mujoco.OffRenderViewer* method), 515
- render() (*mlpro.wrappers.mujoco.RenderViewer* method), 515
- RenderViewer (class in *mlpro.wrappers.mujoco*), 515
- renormalize() (*mlpro.bf.math.normalizers.Normalizer* method), 314
- renormalize\_on\_event() (*mlpro.oa.streams.basics.OATask* method), 422, 432
- Reservoir (class in *mlpro.rl.pool.envs.bglp*), 474
- reset() (*mlpro.bf.ml.basics.Scenario* method), 363
- reset() (*mlpro.bf.ops.ScenarioBase* method), 305
- reset() (*mlpro.bf.streams.models.Sampler* method), 320
- reset() (*mlpro.bf.streams.models.StreamShared* method), 319
- reset() (*mlpro.bf.streams.samplers.min\_wise.SamplerMinWise* method), 448
- reset() (*mlpro.bf.streams.samplers.random.SamplerRND* method), 449
- reset() (*mlpro.bf.streams.samplers.reservoir\_sampling.SamplerReservoir* method), 450
- reset() (*mlpro.bf.streams.samplers.weighted\_random.SamplerWeightedRL* method), 450
- reset() (*mlpro.bf.systems.basics.Controller* method), 343
- reset() (*mlpro.bf.systems.basics.System* method), 349
- reset() (*mlpro.bf.systems.basics.SystemShared* method), 345
- reset() (*mlpro.bf.various.Timer* method), 275
- reset\_actuators() (*mlpro.rl.pool.envs.bglp.BGLP* method), 483
- reset\_levels() (*mlpro.rl.pool.envs.bglp.BGLP* method), 483
- ress (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 478, 481
- retrieve() (*mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.SumSegmentTree* method), 500
- Reward (class in *mlpro.rl.models\_env*), 375
- reward (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 479, 481
- RLDataStoring (class in *mlpro.rl.models\_train*), 391
- RLDataStoringEval (class in *mlpro.rl.models\_train*), 392
- RLScenario (class in *mlpro.rl.models\_train*), 393
- RLScenarioMBInt (class in *mlpro.rl.models\_agents*), 387
- RLTraining (class in *mlpro.rl.models\_train*), 396
- RLTrainingResults (class in *mlpro.rl.models\_train*), 395
- RobotArm3D (class in *mlpro.rl.pool.envs.robotinhtm*), 497
- RobotHTM (class in *mlpro.rl.pool.envs.robotinhtm*), 497
- RockPaperScissors (class in *mlpro.gt.pool.native.games.rockpaperscissors*), 504
- Routing\_3P (class in *mlpro.gt.pool.native.games.routingproblems\_3p*), 507
- run() (*mlpro.bf.ml.basics.Training* method), 367
- run() (*mlpro.bf.mt.Task* method), 301
- run() (*mlpro.bf.mt.Workflow* method), 303
- run() (*mlpro.bf.ops.ScenarioBase* method), 306
- run() (*mlpro.bf.streams.models.StreamTask* method), 325
- run() (*mlpro.bf.streams.models.StreamWorkflow* method), 327
- run\_cycle() (*mlpro.bf.ml.basics.Training* method), 366
- run\_cycle() (*mlpro.bf.ops.ScenarioBase* method), 305
- run\_loop() (*mlpro.bf.mt.Task* method), 301
- run\_on\_event() (*mlpro.bf.mt.Task* method), 301

## S

- `Sampler` (class in `mlpro.bf.streams.models`), 320
- `SamplerMinWise` (class in `mlpro.bf.streams.samplers.min_wise`), 447
- `SamplerReservoir` (class in `mlpro.bf.streams.samplers.reservoir_sampling`), 449
- `SamplerRND` (class in `mlpro.bf.streams.samplers.random`), 448
- `SamplerWeightedRND` (class in `mlpro.bf.streams.samplers.weighted_random`), 450
- `sampling()` (`mlpro.sl.pool.afct.pytorch.PyTorchBuffer` method), 465
- `SARSBuffer` (class in `mlpro.rl.models_env_ada`), 381
- `SARSElement` (class in `mlpro.rl.models_env_ada`), 381
- `save()` (`mlpro.bf.ml.basics.HyperParamTuner` method), 365
- `save()` (`mlpro.bf.ml.basics.TrainingResults` method), 364
- `save()` (`mlpro.bf.various.Persistent` method), 274
- `save()` (`mlpro.gt.native.basics.GTTrainingResults` method), 418
- `save()` (`mlpro.rl.models_train.RLTrainingResults` method), 396
- `save_plots()` (`mlpro.bf.plot.DataPlotting` method), 284
- `Scenario` (class in `mlpro.bf.ml.basics`), 362
- `ScenarioBase` (class in `mlpro.bf.ops`), 304
- `schedule()` (`mlpro.bf.ui.sciui.framework.SciUITooltip` method), 288
- `ScientificObject` (class in `mlpro.bf.various`), 275
- `SciUI` (class in `mlpro.bf.ui.sciui.main`), 285
- `SciUIComponent` (class in `mlpro.bf.ui.sciui.framework`), 288
- `SciUICursor` (class in `mlpro.bf.ui.sciui.framework`), 288
- `SciUIFrame` (class in `mlpro.bf.ui.sciui.framework`), 289
- `SciUIFrameParam` (class in `mlpro.bf.ui.sciui.framework`), 291
- `SciUIRoot` (class in `mlpro.bf.ui.sciui.framework`), 287
- `SciUIScenario` (class in `mlpro.bf.ui.sciui.framework`), 292
- `SciUISharedDB` (class in `mlpro.bf.ui.sciui.framework`), 287
- `SciUISubplot2D` (class in `mlpro.bf.ui.sciui.framework`), 291
- `SciUISubplot3D` (class in `mlpro.bf.ui.sciui.framework`), 291
- `SciUISubplotRoot` (class in `mlpro.bf.ui.sciui.framework`), 290
- `SciUISubplotSaveDLG` (class in `mlpro.bf.ui.sciui.framework`), 290
- `SciUITabCTRL` (class in `mlpro.bf.ui.sciui.framework`), 289
- `SciUITooltip` (class in `mlpro.bf.ui.sciui.framework`), 288
- `SciUIWindow` (class in `mlpro.bf.ui.sciui.framework`), 287
- `SegmentTree` (class in `mlpro.rl.pool.sarsbuffer.PrioritizedBuffer`), 500
- `Sensor` (class in `mlpro.bf.systems.basics`), 342
- `Set` (class in `mlpro.bf.math.basics`), 310
- `set_actions()` (`mlpro.rl.pool.envs.bglp.BGLP` method), 483
- `set_actuator_value()` (`mlpro.bf.systems.basics.Controller` method), 344
- `set_boundaries()` (`mlpro.bf.math.basics.Dimension` method), 310
- `set_broken()` (`mlpro.bf.systems.basics.State` method), 340
- `set_change()` (`mlpro.rl.pool.envs.bglp.Reservoir` method), 475
- `set_cycle_limit()` (`mlpro.bf.ops.ScenarioBase` method), 305
- `set_event_status()` (`mlpro.bf.ui.sciui.framework.SciUICursor` method), 288
- `set_feature_data()` (`mlpro.bf.streams.models.Instance` method), 319
- `set_feature_data()` (`mlpro.bf.systems.basics.State` method), 340
- `set_filename()` (`mlpro.bf.various.Persistent` method), 274
- `set_flush_events()` (`mlpro.bf.ui.sciui.framework.SciUISubplotRoot` method), 290
- `set_id()` (`mlpro.bf.streams.models.Instance` method), 319
- `set_id()` (`mlpro.bf.various.Id` method), 272
- `set_initial()` (`mlpro.bf.systems.basics.State` method), 340
- `set_instances()` (`mlpro.bf.streams.models.StreamShared` method), 320
- `set_label_data()` (`mlpro.bf.streams.models.Instance` method), 319
- `set_latency()` (`mlpro.bf.systems.basics.System` method), 348
- `set_log_level()` (`mlpro.gt.native.basics.GTCoalition` method), 411
- `set_log_level()` (`mlpro.gt.native.basics.GTCompetition` method), 413
- `set_log_level()` (`mlpro.gt.native.basics.GTPlayer` method), 410
- `set_log_level()` (`mlpro.rl.models_agents.Agent` method), 388
- `set_log_level()` (`mlpro.rl.models_agents.MultiAgent` method), 388

- method*), 390
- `set_mode()` (*mlpro.bf.ops.Mode method*), 304
- `set_mode()` (*mlpro.bf.ops.ScenarioBase method*), 305
- `set_name()` (*mlpro.bf.various.Log method*), 272
- `set_name()` (*mlpro.bf.various.PersonalisedStamp method*), 277
- `set_num_instances()` (*mlpro.bf.streams.models.Sampler method*), 320
- `set_options()` (*mlpro.bf.streams.models.Stream method*), 322
- `set_options()` (*mlpro.bf.streams.streams.csv\_file.StreamMLProFile method*), 435
- `set_overall_reward()` (*mlpro.rl.models\_env.Reward method*), 376
- `set_plot_detail_level()` (*mlpro.bf.plot.Plottable method*), 281
- `set_plot_settings()` (*mlpro.bf.plot.Plottable method*), 281
- `set_plot_step_rate()` (*mlpro.bf.plot.Plottable method*), 281
- `set_position()` (*mlpro.bf.math.geometry.Point method*), 316
- `set_predecessors()` (*mlpro.bf.mt.Task method*), 301
- `set_random_seed()` (*mlpro.bf.ml.basics.AWorkflow method*), 362
- `set_random_seed()` (*mlpro.bf.ml.basics.Model method*), 360
- `set_random_seed()` (*mlpro.bf.streams.models.Stream method*), 322
- `set_random_seed()` (*mlpro.bf.streams.streams.rnd10d.StreamMLProRnd10d method*), 437
- `set_random_seed()` (*mlpro.bf.systems.basics.System method*), 349
- `set_random_seed()` (*mlpro.gt.native.basics.GTCoalition method*), 412
- `set_random_seed()` (*mlpro.gt.native.basics.GTCompetition method*), 414
- `set_random_seed()` (*mlpro.gt.native.basics.GTPlayer method*), 410
- `set_random_seed()` (*mlpro.gt.native.basics.GTSolver method*), 409
- `set_random_seed()` (*mlpro.rl.models\_agents.Agent method*), 389
- `set_random_seed()` (*mlpro.rl.models\_agents.MultiAgent method*), 390
- `set_related_set()` (*mlpro.bf.math.basics.Element method*), 311
- `set_success()` (*mlpro.bf.systems.basics.State method*), 340
- `set_terminal()` (*mlpro.bf.systems.basics.State method*), 340
- `set_theta()` (*mlpro.rl.pool.envs.robotinhtm.RobotArm3D method*), 497
- `set_theta()` (*mlpro.rl.pool.envs.robotinhtm.RobotHTM method*), 498
- `set_timeout()` (*mlpro.bf.systems.basics.State method*), 340
- `set_tstamp()` (*mlpro.bf.various.TStamp method*), 275
- `set_value()` (*mlpro.bf.math.basics.Element method*), 311
- `set_value()` (*mlpro.bf.ml.basics.HyperParamDispatcher method*), 358
- `set_value()` (*mlpro.bf.ml.basics.HyperParamTuple method*), 358
- `set_values()` (*mlpro.bf.math.basics.Element method*), 311
- `set_values()` (*mlpro.bf.ml.basics.HyperParamDispatcher method*), 358
- `set_volume_changes()` (*mlpro.rl.pool.envs.bglp.BGLP method*), 483
- `set_weight()` (*mlpro.bf.systems.basics.ActionElement method*), 341
- `setup()` (*mlpro.bf.ml.basics.Scenario method*), 363
- `setup()` (*mlpro.bf.ops.ScenarioBase method*), 305
- `setup()` (*mlpro.bf.streams.models.StreamScenario method*), 328
- `setup()` (*mlpro.bf.systems.basics.DemoScenario method*), 355
- `setup()` (*mlpro.oa.streams.basics.OAScenario method*), 423, 433
- `setup()` (*mlpro.rl.models\_agents.ActionPlanner method*), 386
- `setup_ext()` (*mlpro.rl.models\_agents.RLScenarioMBInt method*), 387
- `setup_sampler()` (*mlpro.bf.streams.models.Stream method*), 323
- `setup_spaces()` (*mlpro.bf.systems.basics.System static method*), 348
- `setup_spaces()` (*mlpro.bf.systems.pool.doublependulum.DoublePendulum static method*), 455
- `setup_spaces()` (*mlpro.bf.systems.pool.doublependulum.DoublePendulum static method*), 460
- `setup_spaces()` (*mlpro.rl.models\_env.Environment static method*), 380
- `setup_spaces()` (*mlpro.rl.models\_env\_ada.EnvModel static method*), 382
- `setup_spaces()` (*mlpro.rl.pool.envs.bglp.BGLP static method*), 481
- `setup_spaces()` (*mlpro.rl.pool.envs.gridworld.GridWorld static method*), 496
- `setup_spaces()` (*mlpro.rl.pool.envs.robotinhtm.RobotHTM static method*), 498
- `setup_spaces()` (*mlpro.wrappers.mujoco.MujocoHandler*

- method), 516
- Shared (class in *mlpro.bf.mt*), 297
- showtip() (*mlpro.bf.ui.sciui.framework.SciUITooltip* method), 288
- Silo (class in *mlpro.rl.pool.envs.bglp*), 475
- sils (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 477, 481
- simulate\_reaction() (*mlpro.bf.systems.basics.FctSTrans* method), 341
- simulate\_reaction() (*mlpro.bf.systems.basics.MultiSystem* method), 354
- simulate\_reaction() (*mlpro.bf.systems.basics.System* method), 350
- SLAdaptiveFunction (class in *mlpro.sl.basics*), 369
- spawn() (*mlpro.bf.math.basics.Set* method), 310
- speed (*mlpro.rl.pool.envs.bglp.Belt* attribute), 473
- start() (*mlpro.bf.ui.sciui.framework.SciUIWindow* method), 288
- start() (*mlpro.bf.ui.sciui.main.SciUI* method), 285
- start\_global\_refresh() (*mlpro.bf.ui.sciui.framework.SciUISharedDB* method), 287
- start\_t() (*mlpro.rl.pool.envs.bglp.Belt* method), 473
- start\_t() (*mlpro.rl.pool.envs.bglp.VacuumPump* method), 471
- State (class in *mlpro.bf.systems.basics*), 340
- state\_from\_mujoco() (*mlpro.bf.systems.basics.System* method), 351
- status (*mlpro.rl.pool.envs.bglp.Actuator* attribute), 469, 470
- Stream (class in *mlpro.bf.streams.models*), 321
- StreamMLProClouds (class in *mlpro.bf.streams.streams.clouds*), 439
- StreamMLProClouds2D4C1000Static (class in *mlpro.bf.streams.streams.clouds*), 440
- StreamMLProClouds2D4C5000Dynamic (class in *mlpro.bf.streams.streams.clouds*), 441
- StreamMLProClouds3D8C10000Dynamic (class in *mlpro.bf.streams.streams.clouds*), 442
- StreamMLProClouds3D8C2000Static (class in *mlpro.bf.streams.streams.clouds*), 441
- StreamMLProCSV (class in *mlpro.bf.streams.streams.csv\_file*), 434
- StreamMLProOutliers (class in *mlpro.bf.streams.streams.point\_outliers*), 438
- StreamMLProRnd10D (class in *mlpro.bf.streams.streams.rnd10d*), 436
- StreamProvider (class in *mlpro.bf.streams.models*), 323
- StreamScenario (class in *mlpro.bf.streams.models*), 328
- StreamShared (class in *mlpro.bf.streams.models*), 319
- StreamTask (class in *mlpro.bf.streams.models*), 324
- StreamWorkflow (class in *mlpro.bf.streams.models*), 326
- sum() (*mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.SumSegmentTree* method), 500
- SumSegmentTree (class in *mlpro.rl.pool.sarsbuffer.PrioritizedBuffer*), 500
- SupplyDemand\_3P (class in *mlpro.gt.pool.native.games.supplydemand\_3p*), 510
- switch\_adaptivity() (*mlpro.bf.ml.basics.AWorkflow* method), 362
- switch\_adaptivity() (*mlpro.bf.ml.basics.Model* method), 360
- switch\_adaptivity() (*mlpro.gt.native.basics.GTCoalition* method), 411
- switch\_adaptivity() (*mlpro.rl.models\_agents.Agent* method), 388
- switch\_adaptivity() (*mlpro.rl.models\_agents.MultiAgent* method), 389
- switch\_adaptivity() (*mlpro.rl.models\_env\_ada.EnvModel* method), 383
- switch\_logging() (*mlpro.bf.ml.basics.Scenario* method), 363
- switch\_logging() (*mlpro.bf.mt.Workflow* method), 302
- switch\_logging() (*mlpro.bf.systems.basics.System* method), 348
- switch\_logging() (*mlpro.bf.various.Log* method), 272
- switch\_logging() (*mlpro.gt.native.basics.GTCoalition* method), 411
- switch\_logging() (*mlpro.gt.native.basics.GTCompetition* method), 413
- switch\_logging() (*mlpro.gt.native.basics.GTPlayer* method), 410
- switch\_logging() (*mlpro.rl.models\_agents.Agent* method), 388
- switch\_logging() (*mlpro.rl.models\_agents.MultiAgent* method), 389
- switch\_logging() (*mlpro.rl.models\_env.EnvBase* method), 378
- switch\_logging() (*mlpro.rl.models\_train.RLScenario* method), 394
- switch\_solver() (*mlpro.gt.native.basics.GTPlayer* method), 411
- System (class in *mlpro.bf.systems.basics*), 346
- SystemShared (class in *mlpro.bf.systems.basics*), 345
- ## T
- t (*mlpro.rl.pool.envs.bglp.BGLP* attribute), 478, 481



- t\_activated** (*mlpro.rl.pool.envs.bglp.Actuator attribute*), 469, 470  
**t\_end** (*mlpro.rl.pool.envs.bglp.Actuator attribute*), 469, 470  
**t\_end\_max** (*mlpro.rl.pool.envs.bglp.VacuumPump attribute*), 471  
**t\_step** (*mlpro.rl.pool.envs.bglp.BGLP attribute*), 478, 481  
**Task** (class in *mlpro.bf.mt*), 300  
**Timer** (class in *mlpro.bf.various*), 275  
**Training** (class in *mlpro.bf.ml.basics*), 365  
**TrainingResults** (class in *mlpro.bf.ml.basics*), 364  
**TransferFunction** (class in *mlpro.bf.physics.basics*), 332  
**TransferFunction\_Routing3P** (class in *mlpro.gt.pool.native.games.routingproblems\_3p*), 505  
**TransferFunction\_SD3P** (class in *mlpro.gt.pool.native.games.supplydemand\_3p*), 508  
**transport** (*mlpro.rl.pool.envs.bglp.BGLP attribute*), 479, 481  
**transport\_t** (*mlpro.rl.pool.envs.bglp.BGLP attribute*), 480, 481  
**tree** (*mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.SegmentTree attribute*), 500  
**TStamp** (class in *mlpro.bf.various*), 275  
**type\_** (*mlpro.rl.pool.envs.bglp.Actuator attribute*), 470  
**type\_** (*mlpro.rl.pool.envs.bglp.Hopper attribute*), 476  
**type\_** (*mlpro.rl.pool.envs.bglp.Silo attribute*), 475, 476
- ## U
- UnitConverter** (class in *mlpro.bf.physics.unitconverter*), 336  
**unlock()** (*mlpro.bf.mt.Shared method*), 298  
**unschedule()** (*mlpro.bf.ui.sciui.framework.SciUITooltip method*), 288  
**update()** (*mlpro.rl.pool.envs.bglp.Belt method*), 473  
**update()** (*mlpro.rl.pool.envs.bglp.BGLP method*), 483  
**update()** (*mlpro.rl.pool.envs.bglp.Reservoir method*), 475  
**update()** (*mlpro.rl.pool.envs.bglp.VacuumPump method*), 472  
**update\_actuators()** (*mlpro.rl.pool.envs.bglp.BGLP method*), 483  
**update\_joint\_coords()** (*mlpro.rl.pool.envs.robotinhtm.RobotArm3D method*), 497  
**update\_levels()** (*mlpro.rl.pool.envs.bglp.BGLP method*), 483  
**update\_parameters()** (*mlpro.bf.math.normalizers.Normalizer method*), 314  
**update\_parameters()** (*mlpro.bf.math.normalizers.NormalizerMinMax method*), 314  
**update\_parameters()** (*mlpro.bf.math.normalizers.NormalizerZTrans method*), 315  
**update\_plot()** (*mlpro.bf.ml.basics.Scenario method*), 363  
**update\_plot()** (*mlpro.bf.mt.Task method*), 302  
**update\_plot()** (*mlpro.bf.plot.Plottable method*), 282  
**update\_plot()** (*mlpro.bf.streams.models.StreamScenario method*), 329  
**update\_plot()** (*mlpro.bf.streams.models.StreamTask method*), 326  
**update\_plot()** (*mlpro.bf.streams.models.StreamWorkflow method*), 327  
**update\_plot()** (*mlpro.bf.systems.basics.DemoScenario method*), 355  
**update\_plot()** (*mlpro.bf.systems.basics.System method*), 352  
**update\_plot()** (*mlpro.rl.models\_agents.MultiAgent method*), 391  
**update\_plot()** (*mlpro.rl.models\_train.RLScenario method*), 395  
**update\_plot()** (*mlpro.rl.pool.envs.bglp.BGLP method*), 484  
**update\_plot()** (*mlpro.rl.pool.envs.gridworld.GridWorld method*), 497  
**update\_priorities()** (*mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.PrioritizedBuffer method*), 499  
**update\_state()** (*mlpro.bf.systems.basics.SystemShared method*), 345  
**update\_theta()** (*mlpro.rl.pool.envs.robotinhtm.RobotArm3D method*), 497
- ## V
- vacs** (*mlpro.rl.pool.envs.bglp.BGLP attribute*), 478, 481  
**VacuumPump** (class in *mlpro.rl.pool.envs.bglp*), 470  
**vol\_cur\_abs** (*mlpro.rl.pool.envs.bglp.Reservoir attribute*), 474, 475  
**vol\_cur\_rel** (*mlpro.rl.pool.envs.bglp.Reservoir attribute*), 474, 475  
**vol\_init\_abs** (*mlpro.rl.pool.envs.bglp.Reservoir attribute*), 474, 475  
**vol\_max** (*mlpro.rl.pool.envs.bglp.Reservoir attribute*), 474, 475
- ## W
- wait\_async\_tasks()** (*mlpro.bf.mt.Async method*), 300  
**wait\_async\_tasks()** (*mlpro.bf.mt.Workflow method*), 303  
**Window** (class in *mlpro.bf.streams.tasks.windows*), 445  
**Workflow** (class in *mlpro.bf.mt*), 302

## Z

`zero_sum()` (*mlpro.gt.native.basics.GTFunction*  
*method*), [406](#)

`zero_sum()` (*mlpro.gt.native.basics.GTPayoffMatrix*  
*method*), [408](#)