mimoCoRB2

Julian Baader

Apr 14, 2025

CONTENTS

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

GETTING STARTED

This section will guide you through the process of installing mimoCoRB2 and running the examples.

1.1 Introduction

1.1.1 What is mimoCoRB2?

mimoCoRB2 (multiple in multiple out configurable ringbuffer manager) provides a central component of each data acquisition system needed to record and preanalyse data from randomly occurring processes. Typical examples are waveform data as provided by detectors common in quantum mechanical measurements, or in nuclear, particle and astro particle physics, e. g. photo tubes, Geiger counters, avalanche photo-diodes or modern SiPMs. The random nature of such processes and the need to keep read-out dead times low requires an input buffer for fast collection of data and an efficient buffer manager delivering a constant data stream to the subsequent processing steps. While a data source feeds data into the buffer, consumer processes receive the data to filter, reduce, analyze or simply visualize the recorded data. In order to optimally use the available resources, multi-core and multi-processing techniques must be applied.

This project originated from an effort to structure and generalize data acquisition for several experiments in advanced physics laboratory courses at Karlsruhe Institute of Technology (KIT) and has been extensively tested with Ubuntu Linux.

1.1.2 What can it do?

Amongst the core features of mimoCoRB2 are:

- multiprocessing safe ringbuffer for NumPy structured arrays
- setup of multiple ringbuffers and workers from configuration files
- templates for common interactions between buffers (importing/exporting, filtering, processing, observing)
- pre built functions for common operations (oscilloscope, histogram, pulse height analysis)
- gui for monitoring and controlling the system

1.2 Examples

mimoCoRB2 comes with a set of examples that can be used to get started quickly. The examples are located in the *examples* directory of the mimoCoRB2 package alongside an explenation of the experiment. Each example is self-contained and can be run independently. The examples stem from experiments currently running at KIT and are meant to be used as a starting point for your own experiments.

CHAPTER

TWO

USER GUIDE

This user guide provides an in-depth description of mimoCoRB2 features required to build your own applications.

2.1 Buffers

Buffers are at the core of mimoCoRB2. Each buffer is defined by

- name
- slot_count
- data_length
- data_dtype

Each buffer is divided into slots, where each slot holds data and metadata. Data is a numpy structured array of shape (data_length,) and dtype data_dtype. Metadata is a numpy structured array of shape (1,) and dtype metadata_dtype.

```
mimoBuffer.metadata_dtype = dtype([('counter', '<i8'), ('timestamp', '<f8'), ('deadtime',
'<f8')])
```

Buffers are implemented in a multiprocessing safe way using shared memory. This means multiple processes can read and write to the same buffer at the same time without any issues.

2.2 Workers

Workers are used for interactions between buffers.

A worker is defined by

- name
- function
- args
- number_of_processes

The name is used as an identifier for the worker. The function is the function that the worker will run with the provided args. The number_of_processes is the number of processes that the worker will run.

2.3 Setup

A setup file is used to define the buffers and workers of the application. It is written in yaml format and divided into three sections.

2.3.1 Buffers

buffer_name:
slot_count: int data_length: int data_dtype:

field_name1: dtype1 field_name2: dtype2 ...

2.3.2 Workers

worker_name:
function: str file: str number_of_processes: int sinks: list[str] sources: list[str] observes: list[str]

2.4 Simple Workers

2.4.1 Importer

class mimocorb2.worker_templates.**Importer**(*io*)

Worker class for importing data from an external generator.

data_example

metadata_example

config

Examples

```
>>> def worker(*mimo_args):
...     importer = Importer(mimo_args)
...     config = importer.config
...     data_example = importer.writer.data_example
...     buffer_name = importer.writer.name
...     def ufunc():
...         for i in range(config['n_events']):
...             data = np.random.normal(size=data_example.shape)
...             yield data
...         yield None
...     importer(ufunc)
```

__call__(*ufunc*)

Start the generator and write data to the buffer.

ufunc must yield data of the same format as the Importer.writer.data_example and yield None at the end. Metadata (counter, timestamp, deadtime) is automatically added to the buffer.

Parameters
ufunc (*Callable*) – Generator function that yields data and ends with None

Return type
None

__init__(*io*)

Checks the setup.

2.4.2 Exporter

class mimocorb2.worker_templates.**Exporter**(*io*)

Worker class for exporting data and metadata.

If provided with an identical sink events will be copied to allow further analysis.

data_example

metadata_example

Examples

```
>>> def worker(*mimo_args):
...     exporter = Exporter(mimo_args)
...     data_example = exporter.reader.data_example
...     buffer_name = exporter.reader.name
...     for data, metadata in exporter:
...         print(data, metadata)
```

__init__(*io*)

Checks the setup.

2.4.3 Filter

class mimocorb2.worker_templates.**Filter**(*io*)

Worker class for filtering data from one buffer to other buffer(s).

Analyze data using ufunc(data) and copy or discard data based on the result.

data_example

metadata_example

config

Examples

```
>>> def worker(*mimo_args):
...     filter = Filter(mimo_args)
...     min_height = filter.config['min_height']
...     def ufunc(data):
...         if np.max(data) > min_height:
...             return True
...         else:
...             return False
...     filter(ufunc)
```

__call__(*ufunc*)

Start the filter and copy or discard data based on the result of ufunc(data).

Parameters
ufunc (*Callable*) – Function which will be called upon the data (Filter.reader.data_example). The function can return:

bool
True: copy data to every sink False: discard data

list[bool] (mapping to the sinks)
True: copy data to the corresponding sink False: dont copy data to the corresponding sink

Return type
None

__init__(*io*)

Checks the setup.

Check that the number of sources, sinks, and observes are correct. Check that the source and sink shapes and dtypes match.

2.4.4 Processor

class mimocorb2.worker_templates.**Processor**(*io*)

Worker class for processing data from one buffer to other buffer(s).

reader

BufferReader object for reading data from the buffer

Type
BufferReader

writers

List of BufferWriter objects for writing data to the buffers

Type
list[BufferWriter]

Examples

```
>>> def worker(*mimo_args):
...     processor = Processor(mimo_args)
...     def ufunc(data):
...         return [data + 1, data - 1]
...     processor(ufunc)
```

__call__(*ufunc*)

Start the processor and process data using ufunc(data).

Parameters
ufunc (*Callable*) – Function which will be called upon the data (Processor.reader.data_example). When the function returns None the data will be discarded. Otherwise the function must return a list of results, one for each sink. If the result is not None it will be written to the corresponding sink.

Return type
None

__init__(*io*)

Checks the setup.

Parameters
mimo_args (*ArgsAlias*) – List of sources, sinks, observes, and config dictionary

2.4.5 Observer

class mimocorb2.worker_templates.**Observer**(*io*)

Worker class for observing data from a buffer.

observer

BufferObserver object for observing data from the buffer

Type
BufferObserver

Examples

```
>>> def worker(*mimo_args):
...     observer = Observer(mimo_args)
...     generator = observer()
...     while True:
...         data, metadata = next(generator)
...         if data is None:
...             break
...         print(data, metadata)
...         time.sleep(1)
```

__call__()

Start the observer and yield data and metadata.

Yields data and metadata from the buffer until the buffer is shutdown.

Yields

- **data** (*np.ndarray, None*) – Data from the buffer
- **metadata** (*np.ndarray, None*) – Metadata from the buffer

Return type
Generator

__init__(*io*)

Checks the setup.

Parameters
mimo_args (*ArgsAlias*) – List of sources, sinks, observes, and config dictionary

2.5 Complex Workers

In case the simple workers are not enough, you can create highly customized workers by interacting with the BufferIO object passed to the worker directly.

2.5.1 BufferIO

class mimocorb2.mimo_worker.**BufferIO**(*sources*, *sinks*, *observes*, *config*)

Collection of buffers for input/output operations.

Object which is passed to each worker process to provide access to the buffers.

sources

Type
list[BufferReader]

sinks

Type
list[BufferWriter]

observes

Type
list[BufferObserver]

config

Type
dict

logger

Type
logging.Logger

shutdown_sinks()

Shutdown all sink buffers.

__getitem__(*key*)

Get the value of a key from the configuration dictionary.

Examples

```
>>> with io.write[0] as (metadata, data):
```

INDEX

Symbols

B

C

D

E

F

I

L

M

O

P

R

S

W