BoboCEP

r3w0p

Jun 26, 2024

CONTENTS

1 Installation **3**
- 1.1 Dependencies . . . 3
- 1.2 Install via pip . . . 3
- 1.3 Build Manually . . . 4
- 1.4 Development . . . 4

2 Getting Started **5**
- 2.1 Complex Event Processing . . . 5
- 2.2 BoboCEP Architecture . . . 5
- 2.3 Quick Start . . . 7
- 2.4 Why “Bobo”? . . . 8
- 2.5 References . . . 8

3 Examples **9**
- 3.1 Simple . . . 9
- 3.2 Advanced . . . 11
- 3.3 Distributed . . . 14
- 3.4 Flask . . . 19

4 Use Cases **23**
- 4.1 Assisted Living . . . 23

5 Phenomena **29**
- 5.1 Patterns . . . 29
- 5.2 Pattern Builder . . . 31
- 5.3 Runs . . . 32

6 Actions **33**
- 6.1 Handlers . . . 33

7 Distributed **35**
- 7.1 Recovery Scenarios . . . 36

8 Publications **39**

9 Glossary **41**
- 9.1 Contiguity . . . 41
- 9.2 Events . . . 41
- 9.3 Phenomenon . . . 42

10 Source Code **43**

10.1 bobocep 43

11 Developer Guide **107**

11.1 Dependencies 107

11.2 Development Tools 107

12 Contributing **111**

Bibliography **113**

Python Module Index **115**

Index **117**

About

BoboCEP is a Complex Event Processing (CEP) engine designed for edge computing in Internet of Things (IoT) systems to facilitate inferential reasoning and decision-making on streaming data. It provides fault tolerance (FT) via the active replication of partially-completed complex events across multiple instances of the software.

License

It is an open source project, as per the Open Source Definition. The source code can be redistributed and/or modified under the terms of the MIT License.

Documentation

Documentation is hosted by Read the Docs, which also provides a PDF version.

CHAPTER

ONE

INSTALLATION

1.1 Dependencies

1.1.1 Python

BoboCEP requires Python 3.9 or later. If you are using Raspberry Pi OS, it should come with this as standard. You can check this using:

```
python --version
```

However, if a suitable version is not installed, then you will need to install it yourself. See here for available Python versions. The following instructions will focus on Python 3.9.16.

Download Python 3.9.16, unpack it, and then enter the directory containing its files.

```
wget https://www.python.org/ftp/python/3.9.16/Python-3.9.16.tar.xz
tar -xf Python-3.9.16.tar.xz
cd Python-3.9.16
```

Next, configure and install.

```
./configure
make -j 4
sudo make altinstall
```

Finally, update pip to the latest version.

```
python3.9 -m pip install --upgrade pip
```

1.2 Install via pip

You can install the latest version of BoboCEP via pip with:

```
pip install BoboCEP
```

1.2.1 virtualenv

If you would like to install BoboCEP in a virtual environment, consider using the following.

```
sudo apt install python3-virtualenv -y

virtualenv venv
source venv/bin/activate
pip install BoboCEP
deactivate
```

1.3 Build Manually

You can also build BoboCEP manually with:

```
git clone https://github.com/r3w0p/bobocep.git BoboCEP
cd BoboCEP
pip install .
```

1.4 Development

If you want to develop BoboCEP, see Developer Guide for more information.

CHAPTER

TWO

GETTING STARTED

2.1 Complex Event Processing

The primary goal of BoboCEP is to take streaming data that enters a system in a serialised and uncorrelated manner (i.e., **simple events**), and detect temporal **patterns** using it to infer the occurrence of some higher-level **phenomenon**. A **complex event** may be generated if a pattern of a phenomenon is fulfilled with relevant data. See Phenomena for more information.

On pattern fulfilment, an **action** may be taken in response which, in turn, leads to the generation of an **action event** representing what happened during action execution and whether it was successfully executed. See Actions for more information.

Complex events and action events are added back into the system's data stream, where they can be used for use in future pattern detection. See the Glossary for definitions of any unfamiliar terminology.

2.1.1 Example

An office may wish to detect the phenomenon of an office fire through various data patterns that could infer its occurrence. Patterns may include:

1. A sharp rise in temperature sensor readings, followed by smoke detection within 1 minute of the rise in temperature.
2. Significant movement away from working spaces and towards the fire exit.
3. Or, it may simply be the pressing of the fire alarm, with no further correlations necessary.

When any of these patterns are fulfilled by the expected data, a number of actions may be triggered as a response: fire alarms sound, sprinklers are activated, and so on.

2.2 BoboCEP Architecture

The architecture of BoboCEP is based on the *information flow processing* (IFP) architecture proposed by [CM2012]. This architecture is extended by enabling state updates to be synchronised across multiple instances of BoboCEP across a network for fault tolerance.

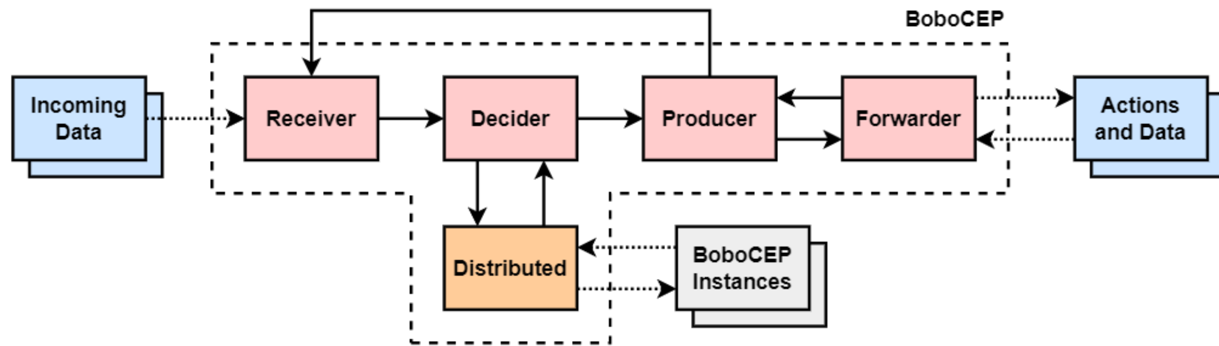

Fig. 1: BoboCEP architecture and external systems. Subsystems within dashed border are the core subsystems for a given BoboCEP instance. Dotted arrows represent data exchange to and from external systems (blue, grey).

2.2.1 Engine

BoboCEP is powered by a BoboEngine instance, which maintains data flow by sequentially executing its IFP subsystems in a round-robin fashion, as follows:

1. The Receiver processes data entering the system, ensuring all data leaving the Receiver is a BoboEvent instance.
2. The Decider checks each event against its patterns: instantiating, halting, advancing, and completing runs as necessary.
3. The Producer generates a BoboEventComplex instance for a newly completed run, which is also sent to the Receiver.
4. The Forwarder executes the complex event's corresponding action and generates a BoboEventAction instance once it has finished, which is also sent to the Receiver.

The subsystems are discussed in more detail below.

Receiver

The Receiver is the entry point for data into the system. Its purpose is to validate incoming data and then format it into a **simple event**. It also consumes **complex events** and **action events** and introduces these event types into the data stream.

- Data of Any type enters BoboReceiver via the add_data method.
- If the data is an instance of type BoboEvent, then it is passed to BoboDecider as-is.
- Otherwise, Receiver will wrap the data in a new BoboEventSimple instance e, where e.data is the data passed via add_data, and will send e to Decider instead.
- A BoboValidator instance is provided to the Receiver to ensure that all data must pass some validation criteria in order to be accepted by the system. For example, BoboValidatorJSONable ensures all data are valid JSON. This validator is essential when using Distributed BoboCEP to ensure all data in the system are serialisable. See Distributed for more information.

Decider

The Decider manages **runs**, which represent instances of **patterns** that require data to fulfil its **predicates**. Once its predicates have been satisfied by a sequence of applicable events, the run is completed.

- A `BoboPhenomenon` contains one or more `BoboPattern` instances and a `BoboAction` instance.
- A `BoboRun` is created when the first predicate of a `BoboPattern` is satisfied with some `BoboEvent`.
- Once a run has had its final predicate satisfied, Producer is notified.

Producer

The Producer generates a **complex event** when it receives notification from the Decider that a run has completed. The complex event is considered as a representation of a phenomenon that has been observed, as per the fulfilment of its pattern via the completed run.

- When a Run has completed, `BoboProducer` will produce a `BoboEventComplex` instance, indicating the observation of the phenomenon that the pattern is evidence of.
- The Producer passes the complex event to Forwarder, in order to trigger the phenomenon's corresponding `BoboActon`. It also passes the complex event to Receiver, which adds it to the data stream.

Forwarder

The Forwarder triggers actions passed to it by Producer and executes them. This may involve communication with external services. Each action leads to the generation of an **action event** that details what occurred during action execution.

- `BoboForwarder` checks the complex event against its phenomenon to determine which action it needs to execute, if any.
- It executes the action and produces a corresponding `BoboEventAction` that contains reference to the complex event which led to its execution.
- The `BoboEventAction` is passed to the Receiver and added to the data stream.

2.2.2 Distributed

`BoboCEP` is able to be executed across multiple software instances and remain synchronised, such that each instance maintains an identical copy of system state through active replication. See Distributed for more information.

2.3 Quick Start

The key components to getting started with `BoboCEP` are as follows.

1. Define the Phenomena that you would like to model by defining one or more patterns per phenomenon. Use `BoboPatternBuilder` for defining patterns to make things much easier.
2. Define Actions that should be executed if a phenomenon were to be triggered. Allocate an action to a phenomenon if you wish, or leave it blank.
3. Decide whether you want BoboCEP to be Distributed or not, and use one of the setup classes to help with setting up the system engine and all of its components: `BoboSetupSimple` and `BoboSetupSimpleDistributed` are provided for these purposes.

Check out the Examples page for various ways to set up BoboCEP and connect it to external systems (e.g., Flask).

To explore the API in more detail, see Source Code.

2.4 Why "Bobo"?

Bobo is the name of Mr Burns' childhood teddy bear that features in the episode "Rosebud" of The Simpsons.

2.5 References

CHAPTER

THREE

EXAMPLES

3.1 Simple

This example demonstrates setting up a simple, non-distributed BoboCEP instance with a single pattern: 1, followed by 2, followed by 3. The engine runs on a separate thread, and a for loop adds numbers 0 to 4 to the data stream in 1-second intervals. When the pattern is fulfilled, an instance of the custom action BoboActionPrint prints "Hello 123!".

```
from threading import Thread
from time import sleep
from typing import Tuple, Any

from bobocep.cep.action import BoboAction, BoboActionHandlerMultithreading
from bobocep.cep.event import BoboEventComplex
from bobocep.cep.phenom import BoboPatternBuilder, BoboPhenomenon, BoboPattern
from bobocep.setup import BoboSetupSimple


class BoboActionPrint(BoboAction):
    """An action that prints a string to stdout."""

    def __init__(self, name: str, message: str):
        """
        :param name: The name of the action.
        :param message: The message to print to stdout.
        """
        super().__init__(name)

        self._message: str = message

    def execute(self, event: BoboEventComplex) -> Tuple[bool, Any]:
        """
        :param event: The complex event generated when my_pattern was
            satisfied with data: 1, 2, 3.

        :return: Whether action execution was successful, followed by
            any additional data (or None).
        """
        print(self._message)

        return True, self._message
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    # A simple pattern to test BoboCEP.
    #
    # The pattern is called "my_pattern" and consists of three predicates.
    # The first predicate checks if an event has data equal to 1.
    # This must be followed by another event with data equal to 2.
    # Finally, a third event must follow with data equal to 3.
    #
    # If three events are input into the BoboCEP system in this order,
    # the pattern is fulfilled and a complex event is generated.
    my_pattern: BoboPattern = BoboPatternBuilder("my_pattern") \
        .followed_by(lambda e, h: int(e.data) == 1) \
        .followed_by(lambda e, h: int(e.data) == 2) \
        .followed_by(lambda e, h: int(e.data) == 3) \
        .generate()

    # The pattern must be associated with a phenomenon which explains what
    # the pattern is trying to model / represent / observe.
    #
    # This phenomenon is called "my_phenomenon".
    #
    # When any of its patterns are fulfilled, its action, BoboActionPrint,
    # will be executed. This action will print a message to stdout.
    my_phenomenon = BoboPhenomenon(
        name="my_phenomenon",
        patterns=[my_pattern],
        action=BoboActionPrint(
            name="my_action",
            message="Hello 123!")
    )

    # The convenience class BoboSetupSimple is used to make BoboCEP setup
    # much simpler. The list of phenomena needs to be provided and an
    # action handler. The handler allows for five actions to be executed
    # concurrently over five threads.
    engine = BoboSetupSimple(
        phenomena=[my_phenomenon],
        handler=BoboActionHandlerMultithreading(threads=5)
    ).generate()

    # BoboCEP is started on a separate thread so that we can pass data to it
    # on the current thread.
    thread_engine = Thread(target=lambda: engine.run())
    thread_engine.start()

    # Data from 0 to 4 are passed to BoboCEP.
    # When 1, 2, 3 are sent, the output will show the action's message.
    for data in range(0, 5):
        print(data)
        engine.receiver.add_data(data)
```

(continues on next page)

(continued from previous page)

```
        sleep(1)

    # The engine and its thread are closed.
    engine.close()
    thread_engine.join()
```

3.2 Advanced

This example shows you how to set up BoboCEP without relying on the BoboSetup classes.

It is the same example as *Simple* above, but with the subsystems of BoboEngine manually assembled.

```
from threading import Thread
from time import sleep
from typing import Tuple, Any

from bobocep.cep.action import BoboAction, BoboActionHandlerMultithreading
from bobocep.cep.engine import BoboEngine
from bobocep.cep.engine.decider import BoboDecider
from bobocep.cep.engine.forwarder import BoboForwarder
from bobocep.cep.engine.producer import BoboProducer
from bobocep.cep.engine.receiver import BoboReceiver
from bobocep.cep.engine.receiver.validator import BoboValidatorAll
from bobocep.cep.event import BoboEventComplex
from bobocep.cep.gen import BoboGenEventIDUnique, BoboGenTimestampEpoch, \
    BoboGenEventTime
from bobocep.cep.phenom import BoboPatternBuilder, BoboPhenomenon, BoboPattern


class BoboActionPrint(BoboAction):
    """An action that prints a string to stdout."""

    def __init__(self, name: str, message: str):
        """
        :param name: The name of the action.
        :param message: The message to print to stdout.
        """
        super().__init__(name)

        self._message: str = message

    def execute(self, event: BoboEventComplex) -> Tuple[bool, Any]:
        """
        :param event: The complex event generated when my_pattern was
            satisfied with data: 1, 2, 3.

        :return: Whether action execution was successful, followed by
            any additional data (or None).
        """
        print(self._message)
```

(continues on next page)

(continued from previous page)

```
        return True, self._message


if __name__ == '__main__':
    # A simple pattern to test BoboCEP.
    #
    # The pattern is called "my_pattern" and consists of three predicates.
    # The first predicate checks if an event has data equal to 1.
    # This must be followed by another event with data equal to 2.
    # Finally, a third event must follow with data equal to 3.
    #
    # If three events are input into the BoboCEP system in this order,
    # the pattern is fulfilled and a complex event is generated.
    my_pattern: BoboPattern = BoboPatternBuilder("my_pattern") \
        .followed_by(lambda e, h: int(e.data) == 1) \
        .followed_by(lambda e, h: int(e.data) == 2) \
        .followed_by(lambda e, h: int(e.data) == 3) \
        .generate()

    # The pattern must be associated with a phenomenon which explains what
    # the pattern is trying to model / represent / observe.
    #
    # This phenomenon is called "my_phenomenon".
    #
    # When any of its patterns are fulfilled, its action, BoboActionPrint,
    # will be executed. This action will print a message to stdout.
    my_phenomenon = BoboPhenomenon(
        name="my_phenomenon",
        patterns=[my_pattern],
        action=BoboActionPrint(
            name="my_action",
            message="Hello 123!")
    )

    # The list of all phenomena under consideration by BoboCEP
    phenomena = [my_phenomenon]

    # Custom URN to prefix before event IDs
    urn = "urn:bobocep:"

    # Accept all data types
    validator = BoboValidatorAll()

    # Unique event ID with custom URN prefixed
    gen_event_id = BoboGenEventIDUnique(urn)

    # Unique run ID with custom URN prefixed
    gen_run_id = BoboGenEventIDUnique(urn)

    # Timestamp is the time since the epoch
    gen_timestamp = BoboGenTimestampEpoch()
```

(continues on next page)

(continued from previous page)

```
# Generate simple event every second and add to Receiver data stream
gen_event = BoboGenEventTime(millis=1000)

# Action handler that can process five actions concurrently
handler = BoboActionHandlerMultithreading(threads=5)

# Create Receiver, where data are first entered into BoboCEP
receiver = BoboReceiver(
    validator=validator,
    gen_event_id=gen_event_id,
    gen_timestamp=gen_timestamp,
    gen_event=gen_event)

# Create Decider, where data are compared against pattern instances (runs)
decider = BoboDecider(
    phenomena=phenomena,
    gen_event_id=gen_event_id,
    gen_run_id=gen_run_id)

# Create Producer, where complex event are generated
producer = BoboProducer(
    phenomena=phenomena,
    gen_event_id=gen_event_id,
    gen_timestamp=gen_timestamp)

# Create Forwarder, where actions are executed and action events generated
forwarder = BoboForwarder(
    phenomena=phenomena,
    handler=handler,
    gen_event_id=gen_event_id,
    gen_timestamp=gen_timestamp)

# Create Engine, which operates the subsystems above
engine = BoboEngine(
    receiver=receiver,
    decider=decider,
    producer=producer,
    forwarder=forwarder)

# BoboCEP is started on a separate thread so that we can pass data to it
# on the current thread.
thread_engine = Thread(target=lambda: engine.run())
thread_engine.start()

# Data from 0 to 4 are passed to BoboCEP.
# When 1, 2, 3 are sent, the output will show the action's message.
for data in range(0, 5):
    print(data)
    engine.receiver.add_data(data)
    sleep(1)

# The engine and its thread are closed.
```

(continues on next page)

(continued from previous page)

```
    engine.close()
    thread_engine.join()
```

If you want to include distributed processing to the above, then add these additional imports:

```
from typing import List
from bobocep.dist import BoboDistributedTCP, BoboDevice
from bobocep.dist.crypto import BoboDistributedCryptoAES
```

Then, add the following, just after creating the `BoboEngine` instance.

```
# Create Device list and AES key accordingly
devices: List[BoboDevice] = ...
aes_key: str = ...

# Generate Distributed TCP instance
distributed: BoboDistributedTCP = BoboDistributedTCP(
    urn=urn,
    decider=engine.decider,
    devices=devices,
    crypto=BoboDistributedCryptoAES(aes_key=aes_key))

# Subscribe Decider to Distributed, and vice versa
engine.decider.subscribe(distributed)
distributed.subscribe(engine.decider)
```

3.3 Distributed

The `BoboSetupSimpleDistributed` class uses TCP for decentralised message-passing, and requires AES encryption to encrypt all traffic between `BoboCEP` instances.

- The AES key that you choose must be 16, 24, or 32 bytes long for AES-128, AES-192, or AES-256 encryption, respectively.

Distributed works by defining a list of `BoboDevice` representing all `BoboCEP` instances (including yourself) that will synchronise together, and providing each device with a unique URN and an ID key to identify the events generated by a device and identify exchanged messages from that device, respectively.

> **Warning:** The ID keys that you use for devices, and the AES key, are expected to be kept secret and **not** hard-coded into your software. The example below is for demonstration purposes only.

Similar to the *Simple* example above, there is a single pattern that expects `a`, followed by `b`, followed by `c`, with an additional haltcondition `h` that will halt any partially-completed run for this pattern.

Firstly, run the following code as `dist_1.py`. This will represent device `urn:bobocep:device:1`.

```
import logging
from threading import Thread, RLock
from typing import Tuple, Any

from flask import Flask
```

(continues on next page)

(continued from previous page)

```
from bobocep.cep.action import BoboAction, BoboActionHandlerMultithreading
from bobocep.cep.engine import BoboEngine
from bobocep.cep.event import BoboEventComplex
from bobocep.cep.phenom import BoboPatternBuilder, BoboPhenomenon, BoboPattern
from bobocep.dist import BoboDevice
from bobocep.setup import BoboSetupSimpleDistributed

app = Flask(__name__)  # v2.2.3
engine: BoboEngine


class BoboActionCounter(BoboAction):
    """
    An action that keeps count of how many times it has been executed,
    and prints the count to stdout each time.

    Note: using RLock with BoboActionHandlerMultiprocessing causes problems,
    so BoboActionHandlerMultithreading is used instead.
    """

    def __init__(self, name: str = "action_counter"):
        """
        :param name: The name of the action.
        """
        super().__init__(name)
        self._lock: RLock = RLock()
        self._count: int = 0

    def execute(self, event: BoboEventComplex) -> Tuple[bool, Any]:
        """
        :param event: The generated complex event.

        :return: Success, and current count.
        """
        with self._lock:
            self._count += 1
            print("{} {}: {}".format(self._name, self._count, event.event_id))

            return True, None


if __name__ == '__main__':
    logging.getLogger().setLevel(logging.DEBUG)

    # A simple pattern: "a" followed by "b" followed by "c".
    # Halt on "h".
    my_pattern: BoboPattern = BoboPatternBuilder("my_pattern") \
        .followed_by(lambda e, h: str(e.data) == "a") \
        .followed_by(lambda e, h: str(e.data) == "b") \
        .followed_by(lambda e, h: str(e.data) == "c") \
        .haltcondition(lambda e, h: str(e.data) == "h") \
```

(continues on next page)

(continued from previous page)

```
    .generate()

# When the pattern is fulfilled, its action, BoboActionCounter,
# increments its internal counter and prints a message to stdout.
my_phenomenon = BoboPhenomenon(
    name="my_phenomenon",
    patterns=[my_pattern],
    action=BoboActionCounter()
)

# A list of the devices that are to be part of your distributed BoboCEP
# network. This list should contain yourself and all other external
# BoboCEP instances.
# - URNs are required to be unique for each device.
# - ID keys are expected to be kept secret, and are used for identifying
#   devices even if their addr / port were to change over time.
devices = [
    # This is Device 1 (you).
    BoboDevice(
        addr="127.0.0.1",
        port=8081,
        urn="urn:bobocep:device:1",
        id_key="id_key_device_1"
    ),
    # This is Device 2.
    BoboDevice(
        addr="127.0.0.1",
        port=8082,
        urn="urn:bobocep:device:2",
        id_key="id_key_device_2"
    )
]

# The convenience class BoboSetupSimpleDistributed is used to make
# distributed BoboCEP setup much simpler.
# - The URN needs to match the URN for the devices representing you
#   in the devices list.
# - The AES, as with device ID keys, is expected to be kept secret.
#   Each BoboCEP instance in the distributed network needs to have the
#   same AES key to be able to encrypt and decrypt messages.
engine, dist = BoboSetupSimpleDistributed(
    phenomena=[my_phenomenon],
    handler=BoboActionHandlerMultithreading(threads=5),
    urn="urn:bobocep:device:1",
    devices=devices,
    aes_key="1234567890ABCDEF"
).generate()

# BoboCEP engine and distributed component are run on separate threads.
thread_engine = Thread(target=lambda: engine.run())
thread_dist = Thread(target=lambda: dist.run())
```

(continues on next page)

(continued from previous page)

```
    # Start both threads.
    thread_engine.start()
    thread_dist.start()
```

Then, run the following code as `dist_2.py`. This will represent device `urn:bobocep:device:2`. This code also has an additional `for` loop that will generate data for `device:2` to consume. If you watch the outlog logs, you should see state synchronisation between both devices.

```
import logging
import time
from threading import Thread, RLock
from typing import Tuple, Any

from flask import Flask

from bobocep.cep.action import BoboAction, BoboActionHandlerMultithreading
from bobocep.cep.engine import BoboEngine
from bobocep.cep.event import BoboEventComplex
from bobocep.cep.phenom import BoboPatternBuilder, BoboPhenomenon, BoboPattern
from bobocep.dist import BoboDevice
from bobocep.setup import BoboSetupSimpleDistributed

app = Flask(__name__)  # v2.2.3
engine: BoboEngine


class BoboActionCounter(BoboAction):
    """
    An action that keeps count of how many times it has been executed,
    and prints the count to stdout each time.

    Note: using RLock with BoboActionHandlerMultiprocessing causes problems,
    so BoboActionHandlerMultithreading is used instead.
    """

    def __init__(self, name: str = "action_counter"):
        """
        :param name: The name of the action.
        """
        super().__init__(name)
        self._lock: RLock = RLock()
        self._count: int = 0

    def execute(self, event: BoboEventComplex) -> Tuple[bool, Any]:
        """
        :param event: The generated complex event.

        :return: Success, and current count.
        """
        with self._lock:
            self._count += 1
            print("{} {}: {}".format(self._name, self._count, event.event_id))
```

(continues on next page)

(continued from previous page)

```
            return True, None


if __name__ == '__main__':
    logging.getLogger().setLevel(logging.DEBUG)

    # A simple pattern: "a" followed by "b" followed by "c".
    # Halt on "h".
    my_pattern: BoboPattern = BoboPatternBuilder("my_pattern") \
        .followed_by(lambda e, h: str(e.data) == "a") \
        .followed_by(lambda e, h: str(e.data) == "b") \
        .followed_by(lambda e, h: str(e.data) == "c") \
        .haltcondition(lambda e, h: str(e.data) == "h") \
        .generate()

    my_phenomenon = BoboPhenomenon(
        name="my_phenomenon",
        patterns=[my_pattern],
        action=BoboActionCounter()
    )

    devices = [
        # This is Device 1.
        BoboDevice(
            addr="127.0.0.1",
            port=8081,
            urn="urn:bobocep:device:1",
            id_key="id_key_device_1"
        ),
        # This is Device 2 (you).
        BoboDevice(
            addr="127.0.0.1",
            port=8082,
            urn="urn:bobocep:device:2",
            id_key="id_key_device_2"
        )
    ]

    # Distributed for Device 2.
    engine, dist = BoboSetupSimpleDistributed(
        phenomena=[my_phenomenon],
        handler=BoboActionHandlerMultithreading(threads=5),
        urn="urn:bobocep:device:2",
        devices=devices,
        aes_key="1234567890ABCDEF"
    ).generate()

    # BoboCEP engine and distributed component are run on separate threads.
    thread_engine = Thread(target=lambda: engine.run())
    thread_dist = Thread(target=lambda: dist.run())
```

(continues on next page)

(continued from previous page)

```
    # Start both threads.
    thread_engine.start()
    thread_dist.start()

    time.sleep(5)

    # Additional code to generate "a", "b", "c" five times.
    for _ in range(5):
        for data in ["a", "b", "c"]:
            engine.receiver.add_data(data)
            time.sleep(3)
```

3.4 Flask

Similar to the *Simple* example above, there is a single pattern that expects 1, followed by 2, followed by 3. However, these values must, instead, be provided via GET requests using a Flask server.

This can be accomplished with three separate calls, as follows:

- `http://127.0.0.1:8080/data/int/1`
- `http://127.0.0.1:8080/data/int/2`
- `http://127.0.0.1:8080/data/int/3`

Each time the phenomenon's pattern is fulfilled, it increments its internal counter and prints a message to stdout, displaying its action name, the current counter value, and the ID of the complex event that was generated. For example: `action_counter 1:  1681645446_0`.

> **Warning:** If you want to use the `BoboActionHandlerMultiprocessing` action handler, then using `RLock` may cause action execution to not work as intended. Therefore, it is recommended that you use the `Blocking` and `Multithreading` handlers for synchronised action execution.

```
from threading import RLock, Thread
from typing import Tuple, Any, Optional
from datetime import datetime
from flask import Flask

from bobocep.cep.action import BoboAction, BoboActionHandlerMultithreading
from bobocep.cep.engine import BoboEngine
from bobocep.cep.event import BoboEventComplex
from bobocep.cep.phenom import BoboPattern, BoboPatternBuilder, \
    BoboPhenomenon
from bobocep.setup import BoboSetupSimple

app = Flask(__name__)  # v2.2.3
engine: Optional[BoboEngine] = None


# A Flask interface that enables integer data to be passed via a GET request.
# For example: 127.0.0.1/data/int/6
```

(continues on next page)

(continued from previous page)

```
@app.route("/data/int/<my_int>", methods=['GET'])
def data_int(my_int):
    global engine
    engine.receiver.add_data(int(my_int))
    return str(int(my_int))


# A Flask interface at index that returns the current time in ISO8601 format.
@app.route("/", methods=['GET'])
def index():
    return datetime.now().isoformat()


class BoboActionCounter(BoboAction):
    """
    An action that keeps count of how many times it has been executed,
    and prints the count to stdout each time.
    """

    def __init__(self, name: str = "action_counter"):
        """
        :param name: The name of the action.
        """
        super().__init__(name)
        self._lock: RLock = RLock()
        self._count: int = 0

    def execute(self, event: BoboEventComplex) -> Tuple[bool, Any]:
        """
        :param event: The generated complex event.

        :return: Success, and current count.
        """
        with self._lock:
            self._count += 1
            print("{} {}: {}".format(self._name, self._count, event.event_id))

            return True, None


if __name__ == '__main__':
    # A simple pattern to test BoboCEP.
    #
    # The pattern is called "my_pattern" and consists of three predicates.
    # The first predicate checks if an event has data equal to 1.
    # This must be followed by another event with data equal to 2.
    # Finally, a third event must follow with data equal to 3.
    #
    # If three events are input into the BoboCEP system in this order,
    # the pattern is fulfilled and a complex event is generated.
    my_pattern: BoboPattern = BoboPatternBuilder("my_pattern") \
        .followed_by(lambda e, h: int(e.data) == 1) \
```

(continues on next page)

(continued from previous page)

```
        .followed_by(lambda e, h: int(e.data) == 2) \
        .followed_by(lambda e, h: int(e.data) == 3) \
        .generate()

    # The pattern must be associated with a phenomenon which explains what
    # the pattern is trying to model / represent / observe.
    #
    # This phenomenon is called "my_phenomenon".
    #
    # When any of its patterns are fulfilled, its action, BoboActionCounter,
    # increments its internal counter and prints a message to stdout.
    my_phenomenon = BoboPhenomenon(
        name="my_phenomenon",
        patterns=[my_pattern],
        action=BoboActionCounter()
    )

    # The convenience class BoboSetupSimple is used to make BoboCEP setup
    # much simpler. The list of phenomena needs to be provided and an
    # action handler. The handler allows for five actions to be executed
    # concurrently over five threads.
    engine = BoboSetupSimple(
        phenomena=[my_phenomenon],
        handler=BoboActionHandlerMultithreading(threads=5)
    ).generate()

    # BoboCEP is started on a separate thread so that we can pass data to it
    # via Flask interface calls.
    thread_engine = Thread(target=lambda: engine.run())
    thread_engine.start()

    # The Flask server is started.
    app.run(
        host="0.0.0.0",
        port=8080,
        debug=True,
        use_reloader=False)
```

CHAPTER FOUR

USE CASES

4.1 Assisted Living

An area of interest in Internet of Things is the need for Smart Home solutions that are designed for people with disability and mobility needs in order to provide independent Assisted Living.

This could be accomplished through sensors that trigger events with minimal or unconventional means of interaction by the end user, such as through voice, controllers, or simply human presence. Actuators, such as plugs, locks, kitchen appliances, and alarms, can trigger as a consequence of these minimal human interactions. Assisted Living can leverage the low-latency event detection and actuation from `BoboCEP` to ensure a reliable and rapid response to events.

Note: The example below is for illustrative purposes only.

4.1.1 Phenomenon

Consider a single phenomenon that one might wish to detect and automatically respond to in an Assisted Living solution. For example, if an elderly resident were to fall in their home or living facility and require immediate assistance, then this 'phenomenon' could be detected through one or more different patterns of correlated, temporal data.

```
phenom_fall = BoboPhenomenon(
    name="Fall",
    patterns=[pattern_1, pattern_2],
    action=action_fall
)
```

Pattern #1

A person is detected entering the living room and they have not been detected moving for at least `60` seconds, nor detected in any other room.

```
from bobocep.cep.phenom import BoboPattern, BoboPatternBuilder

# Living room motion starts the pattern, where event data is a dict that
# contains keys "room" and "motion".
fb1 = lambda e, h: e.data["room"] == "living_room" and e.data["motion"] == 1

# No motion in living room after 60 seconds. The duration between the event
# accepted during fb1 and the event being evaluated with fb2 is used to
```

(continues on next page)

(continued from previous page)

```
# determine duration, where timestamp is milliseconds since the epoch.
fb2 = lambda e, h: e.data["room"] == "living_room" and e.data["motion"] == 0 \
                   and (e.timestamp - h.first().timestamp) / 1000 >= 60

# Any motion in any room halts the pattern.
hc1 = lambda e, h: e.data["motion"] == 1

my_pattern: BoboPattern = BoboPatternBuilder("pattern_1") \
    .followed_by(fb1) \
    .followed_by(fb2) \
    .haltcondition(hc1) \
    .generate()
```

Pattern #2

A heart-rate sensor was operating at healthy levels (for example, between 60 to 140 beats per minute), but has started to consistently report values outside of this range.

```
from bobocep.cep.phenom import BoboPattern, BoboPatternBuilder

# Initial indication that heart-rate sensor operates within normal levels
fb1 = lambda e, h: 60 <= int(e.data) <= 140

# Abnormal heart rates. Using the `times` parameter in the `followed_by`
# method below, 10 events that match fb2 must occur in sequence.
fb2 = lambda e, h: not(60 <= int(e.data) <= 140)

# Halts if still operating within normal levels.
hc1 = lambda e, h: 60 <= int(e.data) <= 140

my_pattern: BoboPattern = BoboPatternBuilder("pattern_1") \
    .followed_by(fb1) \
    .followed_by(fb2, times=10) \
    .haltcondition(hc1) \
    .generate()
```

4.1.2 Actions

On fulfilment of the phenomenon via any of its patterns, a complex event is generated and one or more actions may be triggered.

For this, we could use the `BoboActionMultiSequential` action, which takes multiple actions and runs them sequentially. It can attempt to run them all in sequence and continue execution even if some of them were to fail. This is useful for our scenario because we can trigger several actions for redundancy. For example, we can notify multiple neighbours of the emergency even if some requests failed to send.

```
action_fall = BoboActionMultiSequential(
    name="Action Fall",
    actions=[action_unlock_door, action_notify_neighbours],
    stop_on_fail=False
)
```

Action #1

Unlock the front door, to allow for easy access by neighbours, care workers, or paramedics.

Below is a custom action, `BoboActionIFTTTWebhooks`, that uses the IFTTT Webhooks Integration to accomplish this. A Webhooks request can trigger various smart locks that are integrated into the IFTTT service. For example, Kubu or Nuki.

The `webhooks_event_name` parameter is the custom Event Name that is entered when setting up the Webhooks integration. The `webhooks_key` is provided in the Webhooks Documentation that appears here after making an event.

```
from typing import Tuple, Any
from bobocep.cep.action import BoboAction
from bobocep.cep.event import BoboEventComplex
import requests


class BoboActionIFTTTWebhooks(BoboAction):
    """
    An action that triggers an event using the IFTTT Webhooks integration.
    See: https://ifttt.com/maker_webhooks
    """

    _URL = "https://maker.ifttt.com/trigger/{0}/json/with/key/{1}"

    def __init__(
            self,
            name: str,
            webhooks_event_name: str,
            webhooks_key: str,
            *args,
            **kwargs):
        """
        :param name: The action name.
        :param webhooks_event_name: IFTTT Webhooks event name.
        :param webhooks_key: IFTTT Webhooks key.
        :param args: Action arguments.
        :param kwargs: Action keyword arguments.
        """
        super().__init__(name=name, args=args, kwargs=kwargs)

        self._webhooks_event_name = webhooks_event_name
        self._webhooks_key = webhooks_key

    def execute(self, event: BoboEventComplex) -> Tuple[bool, Any]:
        """
        :param event: The complex event that triggered the action.

        :return: A tuple containing:
                 whether the event request was sent successfully; and
                 the name of the event that was sent.
        """
        response = requests.get(self._URL.format(
            self._webhooks_event_name,
```

(continues on next page)

(continued from previous page)

```
            self._webhooks_key))

        return response.ok, self._webhooks_event_name
```

Action #2

Notify a neighbour via SMS.

Below is a custom action, BoboActionTwilioSMS, that uses the Twilio SMS API to accomplish this. Note: the code below requires the additional twilio package.

```
from typing import Tuple, Any
from bobocep.cep.action import BoboAction
from bobocep.cep.event import BoboEventComplex
from twilio.rest import Client  # https://pypi.org/project/twilio/


class BoboActionTwilioSMS(BoboAction):
    """
    An action that sends an SMS via the Twilio API.
    """

    def __init__(
            self,
            name: str,
            account_sid: str,
            auth_token: str,
            num_from: str,
            num_to: str,
            message: str,
            *args,
            **kwargs):
        """
        :param name: The action name.
        :param account_sid: Twilio Account SID.
        :param auth_token: Twilio Auth Token.
        :param num_from: Twilio phone number.
        :param num_to: Recipient phone number.
        :param message: Message to send to recipient.
        :param args: Action arguments.
        :param kwargs: Action keyword arguments.
        """
        super().__init__(name=name, args=args, kwargs=kwargs)

        self._client = Client(account_sid, auth_token)
        self._num_from = num_from
        self._num_to = num_to
        self._message = message

    def execute(self, event: BoboEventComplex) -> Tuple[bool, Any]:
        """
```

(continues on next page)

(continued from previous page)

```
        :param event: The complex event that triggered the action.

        :return: A tuple containing:
                 whether the SMS was sent successfully; and
                 the recipient phone number.
        """
        message = self._client.messages.create(
            from_=self._num_from,
            body=self._message,
            to=self._num_to
        )

        success = message.status in ("delivered", "queued", "sending", "sent")
        return success, self._num_to
```

Or, notify multiple neighbours with a sequential action.

Each action in `actions` below would be an instance of `BoboActionTwilioSMS` but with differing `num_to` values, which represents the recipient's phone number.

```
action_notify_neighbours = BoboActionMultiSequential(
    name="Notify Neighbours",
    actions=[action_neighbour_1, action_neighbour_2],
    stop_on_fail=False
)
```

CHAPTER FIVE

PHENOMENA

A data stream contains uncorrelated, simple events, e.g. sensor data on a room's temperature at a given point in time.

However, there may exist correlations among sensor data to indicate higher-level **phenomena** occurring in the physical space that simple data events alone cannot identify. Identifying such phenomena would give rise to higher-level **complex events** which, in turn, can be used to detect further complex phenomena, and so on.

Therefore, a phenomenon represents some (real-world) circumstance that one may wish to model in the system by observing **patterns** in streaming data that, when fulfilled with applicable data events, infer the existence or occurrence of the phenomenon at that point in time.

In `BoboCEP`, a phenomenon has one or more patterns whereby, if any of the patterns were fulfilled with appropriate data, then a complex event is generated to represent that the phenomenon was identified at that given point in time, with correlated data events as evidence for this observation.

5.1 Patterns

Patterns are modelled as a series of blocks, with each block containing one or more predicates.

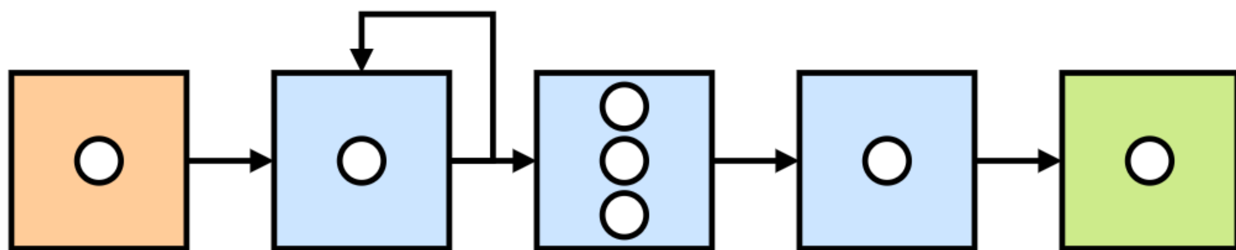

Fig. 1: A series of blocks for a pattern. White circles represent predicates. The starting block is orange, intermediary blocks are blue, and the final block is green.

Fulfilling a pattern starts with an event that satisfies the predicate of the first block of the pattern. From there, a **run** is generated to track the progress across the blocks of the pattern (see *Runs*).

Checking an event against a predicate is as follows.

```
def evaluate(self, event: BoboEvent, history: BoboHistory) -> bool:
    ...
```

During evaluation, the `event` being checked is passed, as well as the `history` of all events that were previously accepted by the previous blocks in the series. If any predicate in a block evaluates to `True`, the event is accepted and added to the history, and the block moves to the next block in the series.

Each block can have a group name associated with it, which will be used in the `history` to group accepted events. Multiple blocks can share the same group name.

Blocks can have the following additional properties.

5.1.1 Negated

Negated blocks can check whether an event does *not* pass its predicate i.e. predicate success is based on whether it returns `False` instead of `True`. First and final blocks cannot be negated. A negated block cannot also be optional (see next).

5.1.2 Optional

Optional blocks may be satisfied by an event but, if not, then the event is also checked against the subsequent block. If that block is also optional, it continues on until either an optional block is satisfied by the event, or a non-optional block is reached. First and final blocks cannot be optional. As stated, an optional block cannot also be negated.

5.1.3 Looping

Looping blocks enable both the current block and next block to be potential paths through a pattern's block series. First and final blocks cannot loop. A looping block can neither be negated nor optional.

5.1.4 Contiguity

If an event is checked against a block and is **not** accepted, then contiguity determines what should happen next.

- **Strict** contiguity means that the pattern should **halt** (i.e. stop) and all progress towards the final block is lost. A strict block cannot also be optional.
- **Relaxed** contiguity means that the pattern can tolerate events entering the system which it does not require for its pattern.

5.1.5 Conditions

In addition to blocks, there are also **preconditions** and **haltconditions**. These are additional predicates against which an event is evaluated *before* passing the event onto the current block's predicate(s).

- **Preconditions** are predicates whereby, if an event does not successfully match against *all* predicates, then the pattern will halt. If it does match against them all, then the event will be passed to the current block of the pattern. For example, a precondition may be that all data originates from a single IP address.
- **Haltconditions** are predicates whereby, if an event successfully matches against *any* predicate, then the pattern will halt. If it does not match any haltcondition, then the event will be passed to the current block of the pattern. For example, a haltcondition may be to halt if 60 seconds has passed since the first event in the history (i.e., the pattern must reach completion within 60 seconds).

Note: Preconditions will cause a pattern to halt if it encounters *any* event that does not satisfy the predicates of all preconditions. That is, preconditions provide strict contiguity. Unless you are also using strict contiguity in all of your patterns, it may be best to avoid using preconditions.

5.2 Pattern Builder

Creating a pattern is best achieved using the `BoboPatternBuilder`.

```
from bobocep.cep.phenom import BoboPatternBuilder

builder = BoboPatternBuilder(name="my_pattern")
```

The constructor requires a `name` for the pattern's name, and can optionally have its `singleton` parameter set to `True` (the default is `False`).

```
builder = BoboPatternBuilder(name="my_pattern", singleton=True)
```

Setting `singleton` to `True` means that only one run for this pattern can be active at any given time. For a new run to be instantiated, the existing run must first be completed or halted. If it is `False`, then an unlimited number of runs can be created from the pattern.

The pattern builder uses various methods to determine the flow of the pattern from one block to another, specifying the predicates and contiguity along the way. The methods are as follows.

- Methods `next` and `not_next` are used for strict contiguity and negated strict contiguity, respectively;
- Methods `followed_by` and `not_followed_by` for relaxed contiguity;
- Methods `followed_by_any` and `not_followed_by_any` for non-deterministic relaxed contiguity;
- Methods `precondition` and `haltcondition` to provide predicates accordingly.

For example, calling `next` on the pattern builder means that the block being added to the pattern contains a predicate that must be satisfied by the *very next event* that enters the system. If this event does not satisfy, the run is halted. The `followed_by` method adds a block that will wait until any future event satisfies it. For most applications, `followed_by` will be the most suitable choice.

```
builder.followed_by(
    predicate=lambda e, h: type(e.data) == int and e.data == 15,
    group="my_group",
    times=3,
    loop=False,
    optional=False
)
```

In the example above, predicate `lambda e, h` is a function consisting of event `e` to check and the current history `h` of all previous events accepted by the run. Event `e` is a subtype of `BoboEvent` and `h` of type `BoboHistory`.

Additionally, optional arguments have been provided:

- A group name `my_group` in which the history will store this event, should an event be accepted by this predicate.
- The `times` option adds three blocks, in series, to the pattern, all with identical characteristics. That is, The predicate will need to be satisfied 3 times by 3 separate events.
- The 3 blocks are not self-looping.
- The 3 blocks are not optional.

```
from bobocep.cep.phenom import BoboPattern

pattern: BoboPattern = builder.generate()
```

5.3 Runs

Runs serve as instances of patterns. Each pattern can have multiple runs at any given time, for example, if the first predicate of the pattern is satisfied multiple times.

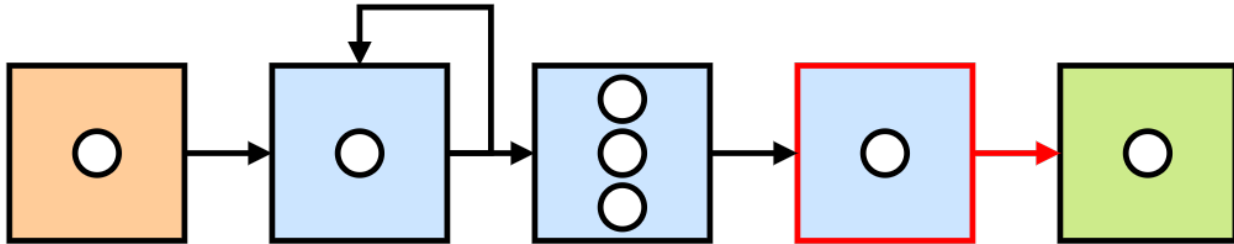

Fig. 2: A run is as an instance of a pattern that keeps track of its state across the pattern's blocks. The run's current block is indicated in red. For this run to complete, it must be passed an event that satisfies the predicate in the final block (green).

Runs work as follows:

1. When the first predicate of a pattern has been satisfied by an event, a run is **generated**.
2. The run continues to monitor the state of the partially-completed pattern as more and more events 'push' the currently-monitored block towards the pattern's final block.
3. Once the final block's predicate has been satisfied, the Producer is notified of the completed run, leading to the Producer generating a **complex event** which is sent to the Receiver.
4. The Forwarder, in turn, executes the associated phenomenon's Action (if one exists). Once it has finished execution, whether successful or not, an **action event** is produced and sent to the Receiver.

If a run needs to end before reaching the final state (e.g., because of a contiguity requirement or satisfied haltcondition), then it enters a **halted** state and is removed from the list of active runs.

CHAPTER SIX

ACTIONS

On the completion of a pattern's run, the Producer is notified and produces a complex event in response, which represents the detection of the pattern's phenomenon.

The Producer then notifies the Forwarder that the phenomenon's action should be executed in response.

```
def execute(self, event: BoboEventComplex) -> Tuple[bool, Any]:
    ...
```

On action execution, the `execute` function is provided with a copy of the complex event, and is expected to return two things:

1. Whether the action was successful or not: `True` or `False`, accordingly.
2. Any additional data, or `None`. Note that, if using Distributed `BoboCEP`, the data type should be JSONable. See Distributed for more information.

6.1 Handlers

In `BoboCEP`, Forwarder contains an **action handler** which is responsible for executing actions and passing the action's response back. The Forwarder then generates an action event which is sent to Receiver.

Note: In Distributed `BoboCEP`, only the instance that first completes a run will be the instance that handles the action.

The default action handlers provided by `BoboCEP` are as follows.

6.1.1 Blocking

The blocking handler blocks the thread on which `BoboCEP` is running on while it executes its actions. This is useful when `BoboCEP` is required to execute actions deterministically, *one action at a time*, in the order that they are sent to Forwarder.

```
from bobocep.cep.action import BoboActionHandlerBlocking

handler = BoboActionHandlerBlocking()
```

6.1.2 Multithreading

The multithreading handler uses `n` threads to execute actions concurrently.

```
from bobocep.cep.action import BoboActionHandlerMultithreading

handler = BoboActionHandlerMultithreading(threads=5)
```

6.1.3 Multiprocessing

The multiprocessing handler utilises multicore processing by specifying uses `n` processes on which to execute actions simultaneously.

```
from bobocep.cep.action import BoboActionHandlerMultiprocessing
from multiprocessing import cpu_count

# Processes equal to one less than the maximum system CPUs available.
handler = BoboActionHandlerMultiprocessing(processes=max(1, cpu_count() - 1))
```

CHAPTER

SEVEN

DISTRIBUTED

BoboCEP is able to be distributed over multiple devices for fault-tolerant CEP at the network edge. Each software instance is able to synchronise with one-another in a decentralised manner.

It accomplishes this by sharing any changes that occur to runs locally, and updating local runs when remote instances notify it of its updates. The type of message sent depends on the last successful communication that an instance has had with an external instance.

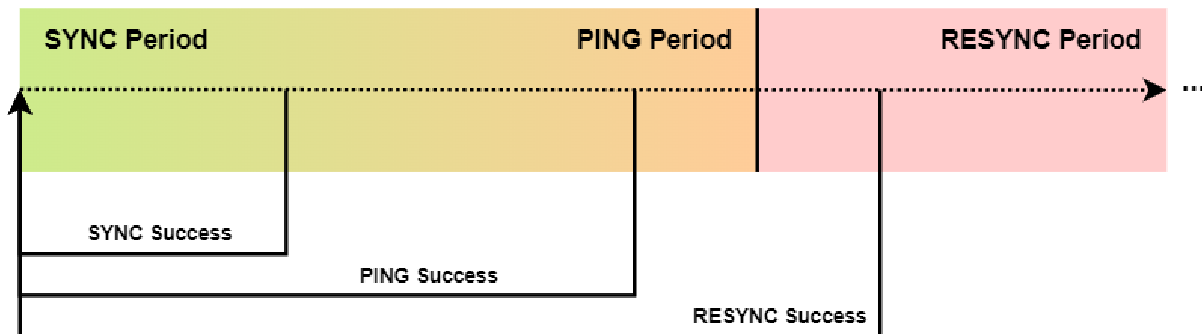

Fig. 1: Distributed BoboCEP sends different message types depending on the period of time since last communication with an external instance.

- SYNC requests sync the most recent run changes: whether a run has completed, halted, or had its internal state updated with a new event.
- PING requests simply ping the other instance if it has not sent a SYNC request in a while, to make sure the other instance is still online.
- RESYNC requests are sent when an instance has not been able to SYNC or PING another for a while, and ensures that the remote instance has the full system state. These requests may fail several times before being successful, depending on whether the external instance ever comes back online or not, or whether it recovers from its network failure, for example.

Additionally, a RESET flag may be sent with messages to indicate to other instances that they should fully RESYNC with the instance that sent the flag. This is useful when a hardware or software crash requires an instance gets back up to speed; BoboCEP does not store internal state to disk.

Note: Distributed BoboCEP is designed to be deployed at the edge of a single network, ideally with only 2-3 software instances.

Distributed BoboCEP provides the additional benefit of being able to **load-balance** data input into the system, by simply delegating sensors to different BoboCEP instances and having them exchange their run changes accordingly.

For Distributed BoboCEP to operate correctly, all data in the system must be serialisable. This can be enforced by using an appropriate validator for the Receiver. For example, BoboValidatorJSONable ensures that all data are valid JSON, and this is the required validator for the BoboSetupSimpleDistributed setup class. See Examples for how to use this setup class.

7.1 Recovery Scenarios

BoboCEP is designed to handle errors and discrepancies with distributed processing in the least complex way possible, that does not rely on excessive message passing as part of its recovery strategy. Various scenarios, and their expected recovery strategies, are discussed below.

The scenarios below consider three distributed instances - A, B, and C - that are hosted on three separate devices.

7.1.1 Communication Failure

Cannot send data to another instance.

If some instance A is unable to communicate with B, then it will store all of the SYNC data that it has been unable to send to B in a data **"stash"** and will periodically reattempt sending the stash in the future.

If B enters the PING period, it will still attempt to send SYNC data if there is any in the stash or if a run is completed, halted, or updated in Decider in the meantime. If B enters the RESYNC period, its stash is wiped and it will be forced to fully resynchronise with all runs in the Decider in A.

Note: Decider is able to cache its n most recently completed and halted runs. These cached runs are sent during a RESYNC.

7.1.2 Run Complete

Multiple instances complete same run with different events.

In this scenario, the complex events produced by the different instances will have different histories: where the final event accepted by the respective runs will vary.

For simplicity, BoboCEP will not attempt to rectify the histories. This is because:

- It is not possible to determine which history should be the one to keep; and
- Inconsistent events will still be mostly comparable, in the sense that the accepted events must have satisfied the same predicate. For example: if the predicate requires a temperature between 0 and 5 Celsius, then all events must have data within that acceptable range anyway.

7.1.3 Run Halt

One instance completes run, another halts it.

For example:

- `A` receives Run `r` from `B` which states that it is `COMPLETE`; *however*
- `A` *also* receives Run `r` from `C` which states that it is `HALTED`.

In this scenario, `COMPLETE` takes precedent over `HALT`, and `A` will complete the run, producing a complex event accordingly. It will do this if the run is in progress, or if it has previously been halted by `A`.

It will **not** produce multiple complex events for the same run if it had already been completed by `A` previously.

7.1.4 Run Update

Update is multiple blocks ahead of local run.

For example, `A` receives an `UPDATE` for a run, where the update is several blocks ahead of where its local copy of the run is.

In this scenario, the local run is simply pushed forward to the new block, and the event history of the update replaces the local run history.

Update is behind local run.

In this scenario, the update is ignored.

One instance updates run, another halts it.

In this scenario, `HALT` takes precedent over `UPDATE`, and the local version of the run is `HALTED`.

CHAPTER

EIGHT

PUBLICATIONS

CHAPTER

NINE

GLOSSARY

9.1 Contiguity

The policy for how BoboCEP should respond when it is unable to match an event to a *Run*.

9.1.1 Strict

All matching events are *strictly* one after the other, without any non-matching events in-between. If an event does not match, the *Run* halts.

9.1.2 Relaxed

Any non-matching events are ignored.

9.1.3 Non-Deterministic Relaxed

The same as relaxed, but allows multiple matches from a state when its transition is non-deterministic.

9.2 Events

9.2.1 Simple Event

Represents primitive data that has entered the BoboCEP system via the Receiver from an external source.

9.2.2 Complex Event

Represents the inference of some phenomenon that was identified by a pattern in other events.

9.2.3 Action Event

Represents the execution of an action by the system and the effect of its execution, if any.

9.2.4 Event History

The events that were accepted by a pattern as being indicative of the existence of a complex event.

9.3 Phenomenon

An observable (real-world) circumstance which, when satisfied by patterns of events, facilitates the generating of a complex event that models the occurrence of the phenomenon.

9.3.1 Pattern

A sequence of data correlations that, when fulfilled with data from a data stream, infer the existence of a complex event.

9.3.2 Run

An instance of a pattern.

In Progress

A run that has been started by consuming its first matching event, but has yet to halt or complete. A run in progress is partially completed and therefore will not produce a complex event until it has completed.

Halted

A run that has stopped because of an event that triggered it to halt. A halted run will not produce a complex event.

Completed

A run that has stopped because its pattern was fully satisfied with events. A completed run will produce a complex event.

CHAPTER TEN

SOURCE CODE

Explore the `BoboCEP` source code below by clicking through the modules. You can also use the `Search docs` tool in the navigation bar to search by name.

Modules

bobocep	Top-level package for BoboCEP.

10.1 bobocep

Top-level package for BoboCEP.

Modules

bobocep.bobocep	Core classes.
bobocep.cep	CEP imports.
bobocep.dist	Distributed imports.
bobocep.setup	Setup imports.

10.1.1 bobocep.bobocep

Core classes.

Classes

BoboJSONable()	An abstract interface for JSONable types.

bobocep.bobocep.BoboJSONable

class bobocep.bobocep.**BoboJSONable**

Bases: ABC

An abstract interface for JSONable types.

__init__()

abstract static from_json_dict(*d: dict*) → *BoboJSONable*

Parameters
d – A JSON *dict* representation of an object of this type.

Returns
A new instance of this type.

abstract static from_json_str(*j: str*) → *BoboJSONable*

Parameters
j – A JSON *str* representation of an object of this type.

Returns
A new instance of this type.

abstract to_json_dict() → dict

Returns
A JSON *dict* representation of an object of this type.

abstract to_json_str() → str

Returns
A JSON *str* representation of an object of this type.

Exceptions

BoboError	A *BoboCEP* error.
BoboJSONableError	A JSONable error.

bobocep.bobocep.BoboError

exception bobocep.bobocep.**BoboError**

A *BoboCEP* error.

bobocep.bobocep.BoboJSONableError

exception bobocep.bobocep.**BoboJSONableError**

A JSONable error.

10.1.2 bobocep.cep

CEP imports.

Modules

bobocep.cep.action	Action imports.
bobocep.cep.engine	Engine imports.
bobocep.cep.event	Event imports.
bobocep.cep.gen	Generator imports.
bobocep.cep.phenom	Phenomenon imports.

bobocep.cep.action

Action imports.

Modules

bobocep.cep.action.action	Action definitions.
bobocep.cep.action.common	Common actions.
bobocep.cep.action.handler	Handlers that coordinate the execution of actions.

bobocep.cep.action.action

Action definitions.

Classes

BoboAction(name, *args, **kwargs)	An action.

bobocep.cep.action.action.BoboAction

class bobocep.cep.action.action.**BoboAction**(*name: str*, **args*, ***kwargs*)

Bases: ABC

An action.

__init__(*name: str*, **args*, ***kwargs*)

Parameters

- **name** – The action name.
- **args** – Action arguments.
- **kwargs** – Action keyword arguments.

abstract execute(*event:* BoboEventComplex) → Tuple[bool, Any]

Parameters
event – The complex event that triggered the action.

Returns
A tuple containing: whether the action execution was successful; and any additional data.

property name: str

Returns
Action name.

Exceptions

BoboActionError	An action error.

bobocep.cep.action.action.BoboActionError

exception bobocep.cep.action.action.**BoboActionError**
An action error.

bobocep.cep.action.common

Common actions.

Modules

bobocep.cep.action.common.multi	Multi actions.

bobocep.cep.action.common.multi

Multi actions.

Classes

BoboActionMulti(name, *args, **kwargs)	An abstract multi action.
BoboActionMultiSequential(name, actions, ...)	An abstract sequential multi action.

bobocep.cep.action.common.multi.BoboActionMulti

class bobocep.cep.action.common.multi.**BoboActionMulti**(*name: str*, **args*, ***kwargs*)

Bases: *BoboAction*, ABC

An abstract multi action.

__init__(*name: str*, **args*, ***kwargs*)

Parameters

- **name** – The action name.
- **args** – Action arguments.
- **kwargs** – Action keyword arguments.

abstract execute(*event:* BoboEventComplex) → Tuple[bool, Any]

Parameters
event – The complex event that triggered the action.

Returns
A tuple containing: whether the action execution was successful; and any additional data.

property name: str

Returns
Action name.

bobocep.cep.action.common.multi.BoboActionMultiSequential

class bobocep.cep.action.common.multi.**BoboActionMultiSequential**(*name: str*, *actions: List[*BoboAction*]*, *stop_on_fail: bool*, **args*, ***kwargs*)

Bases: *BoboActionMulti*

An abstract sequential multi action.

__init__(*name: str*, *actions: List[*BoboAction*]*, *stop_on_fail: bool*, **args*, ***kwargs*)

Parameters

- **name** – The action name.
- **actions** – The list of actions to execute.
- **stop_on_fail** – If True, the multi-action stops processing its action list if its current action fails. If False, it continues to process its remaining actions. Note: failure of any action in its list will cause the multi-action's success to be False.
- **args** – Action arguments.
- **kwargs** – Action keyword arguments.

execute(*event:* BoboEventComplex) → Tuple[bool, List[Tuple[bool, Any]]]

Parameters
event – The complex event that triggered the action.

Returns
A tuple containing: whether all actions were successful; and a list of the output from each individual action.

`property name: str`

Returns
Action name.

bobocep.cep.action.handler

Handlers that coordinate the execution of actions.

Classes

BoboActionHandler([max_size])	An abstract action handler.
BoboActionHandlerBlocking([max_size])	An action handler that blocks during action execution.
BoboActionHandlerMultiprocessing(processes)	An action handler that uses multiprocessing for action execution.
BoboActionHandlerMultithreading(threads[, ...])	An action handler that uses multithreading for action execution.
BoboHandlerResponse(action_name, ...)	A handler response to action execution.

bobocep.cep.action.handler.BoboActionHandler

`class bobocep.cep.action.handler.BoboActionHandler(max_size: int = 0)`

Bases: `ABC`

An abstract action handler.

`__init__(max_size: int = 0)`

Parameters
max_size – Maximum queue size. Default: 0 (unbounded).

`close()` → None

Close the handler.

`get_handler_response()` → *BoboHandlerResponse* | None

Returns
Action response from queue, or *None* if queue is empty.

`handle(action: BoboAction, event: BoboEventComplex)` → Any

Handle an action.

Parameters

- **action** – The action to handle.
- **event** – The complex event that caused the action to trigger.

Returns
A return value from handling the action.

is_closed() → bool

Returns
True if handler is closed; *False* otherwise.

size() → int

Returns
The size of the handler queue.

bobocep.cep.action.handler.BoboActionHandlerBlocking

class bobocep.cep.action.handler.**BoboActionHandlerBlocking**(*max_size: int = 0*)

Bases: *BoboActionHandler*

An action handler that blocks during action execution.

__init__(*max_size: int = 0*)

Parameters
max_size – Maximum queue size. Default: 0 (unbounded).

close() → None

Close the handler.

get_handler_response() → *BoboHandlerResponse* | None

Returns
Action response from queue, or *None* if queue is empty.

handle(*action:* BoboAction, *event:* BoboEventComplex) → Any

Handle an action.

Parameters

- **action** – The action to handle.
- **event** – The complex event that caused the action to trigger.

Returns
A return value from handling the action.

is_closed() → bool

Returns
True if handler is closed; *False* otherwise.

size() → int

Returns
The size of the handler queue.

bobocep.cep.action.handler.BoboActionHandlerMultiprocessing

class bobocep.cep.action.handler.**BoboActionHandlerMultiprocessing**(*processes: int*, *max_size: int = 0*)

Bases: `BoboActionHandler`

An action handler that uses multiprocessing for action execution.

__init__(*processes: int*, *max_size: int = 0*)

Parameters

- **processes** – Number of multicore processes to use for handling actions.
- **max_size** – Maximum queue size. Default: 0 (unbounded).

close() → None

Close the handler.

get_handler_response() → *BoboHandlerResponse* | None

Returns

Action response from queue, or *None* if queue is empty.

handle(*action:* BoboAction, *event:* BoboEventComplex) → Any

Handle an action.

Parameters

- **action** – The action to handle.
- **event** – The complex event that caused the action to trigger.

Returns

A return value from handling the action.

is_closed() → bool

Returns

True if handler is closed; *False* otherwise.

join() → None

Join the multiprocessing pool.

size() → int

Returns

The size of the handler queue.

bobocep.cep.action.handler.BoboActionHandlerMultithreading

class bobocep.cep.action.handler.**BoboActionHandlerMultithreading**(*threads: int*, *max_size: int = 0*)

Bases: `BoboActionHandler`

An action handler that uses multithreading for action execution.

__init__(*threads: int*, *max_size: int = 0*)

Parameters

- **threads** – Number of thread processes to use for handling actions.

- **max_size** – Maximum queue size. Default: 0 (unbounded).

close() → None

Close the handler.

get_handler_response() → *BoboHandlerResponse* | None

Returns
Action response from queue, or *None* if queue is empty.

handle(*action: BoboAction*, *event: BoboEventComplex*) → Any

Handle an action.

Parameters

- **action** – The action to handle.
- **event** – The complex event that caused the action to trigger.

Returns
A return value from handling the action.

is_closed() → bool

Returns
True if handler is closed; *False* otherwise.

join() → None

Join the multiprocessing pool.

size() → int

Returns
The size of the handler queue.

bobocep.cep.action.handler.BoboHandlerResponse

class bobocep.cep.action.handler.**BoboHandlerResponse**(*action_name: str*, *complex_event: BoboEventComplex*, *success: bool*, *data: Any*)

Bases: NamedTuple

A handler response to action execution.

__init__()

action_name: str

Alias for field number 0

complex_event: ***BoboEventComplex***

Alias for field number 1

count(*value*, */*)

Return number of occurrences of value.

data: Any

Alias for field number 3

index(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

success: bool

Alias for field number 2

Exceptions

BoboActionHandlerError	An action handler error.

bobocep.cep.action.handler.BoboActionHandlerError

exception bobocep.cep.action.handler.**BoboActionHandlerError**

An action handler error.

bobocep.cep.engine

Engine imports.

Modules

bobocep.cep.engine.decider	Decider imports.
bobocep.cep.engine.engine	CEP engine.
bobocep.cep.engine.forwarder	Forwarder imports.
bobocep.cep.engine.producer	Producer imports.
bobocep.cep.engine.receiver	Receiver imports.
bobocep.cep.engine.task	CEP engine tasks.

bobocep.cep.engine.decider

Decider imports.

Modules

bobocep.cep.engine.decider.decider	Engine task that detects patterns in data and decides whether a complex event has manifest.
bobocep.cep.engine.decider.pubsub	Decider publish-subscribe classes.
bobocep.cep.engine.decider.run	A run.
bobocep.cep.engine.decider.runserial	A serializable representation of a run.

bobocep.cep.engine.decider.decider

Engine task that detects patterns in data and decides whether a complex event has manifest.

Classes

BoboDecider(phenomena, gen_event_id, gen_run_id)	A decider task.

bobocep.cep.engine.decider.decider.BoboDecider

class bobocep.cep.engine.decider.decider.**BoboDecider**(*phenomena: List[BoboPhenomenon], gen_event_id: BoboGenEventID, gen_run_id: BoboGenEventID, max_cache: int = 0, max_size: int = 0*)

Bases: *BoboEngineTask*, *BoboDeciderPublisher*, *BoboReceiverSubscriber*, *BoboDistributedSubscriber*

A decider task.

__init__(*phenomena: List[BoboPhenomenon], gen_event_id: BoboGenEventID, gen_run_id: BoboGenEventID, max_cache: int = 0, max_size: int = 0*)

Parameters

- **phenomena** – List of phenomena.
- **gen_event_id** – Event ID generator.
- **gen_run_id** – Run ID generator.
- **max_cache** – Max cache size (<=0 means no caching). Default: 0.
- **max_size** – Max queue size. Default: 0 (unbounded).

all_runs() → Tuple[*BoboRun*, ...]

Returns
All active runs in the decider.

close() → None

Closes the Decider.

is_closed() → bool

Returns
True if decider is set to close; *False* otherwise.

on_distributed_update(*completed: List[BoboRunSerial], halted: List[BoboRunSerial], updated: List[BoboRunSerial]*) → None

Parameters

- **completed** – Completed runs.
- **halted** – Halted runs.
- **updated** – Updated runs.

`on_receiver_update`(*event:* BoboEvent) → None

Parameters
`event` – Event from Receiver.

`phenomena`() → Tuple[*BoboPhenomenon*, ...]

Returns
All phenomena under consideration by the decider.

`run_at`(*phenomenon_name: str*, *pattern_name: str*, *run_id: str*) → *BoboRun* | None

Parameters

- `phenomenon_name` – A phenomenon name.
- `pattern_name` – A pattern name.
- `run_id` – A run ID.

Returns
A run associated with the given phenomenon and pattern name; or None if no such run exists.

`runs_from`(*phenomenon_name: str*, *pattern_name: str*) → Tuple[*BoboRun*, ...]

Parameters

- `phenomenon_name` – A phenomenon name.
- `pattern_name` – A pattern name.

Returns
The runs associated with the given phenomenon and pattern name.

`size`() → int

Returns
The total number of events in the decider's queue.

`snapshot`() → Tuple[List[*BoboRunSerial*], List[*BoboRunSerial*], List[*BoboRunSerial*]]

A snapshot of the current state of the Decider.

Returns
Tuple of cached completed, cached halted, and currently partially-completed runs. If caching is disabled, the first two lists will be empty.

`subscribe`(*subscriber:* BoboDeciderSubscriber) → None

Parameters
`subscriber` – Subscriber to Decider data.

`update`() → bool

Performs an update cycle of the decider that takes an event from its queue and checks it against phenomena and existing runs.

Returns
True if an internal state change occurred during the update; False otherwise.

Exceptions

BoboDeciderError	A decider task error.

bobocep.cep.engine.decider.decider.BoboDeciderError

exception bobocep.cep.engine.decider.decider.**BoboDeciderError**

A decider task error.

bobocep.cep.engine.decider.pubsub

Decider publish-subscribe classes.

Classes

BoboDeciderPublisher()	A decider publisher interface.
BoboDeciderSubscriber()	A decider subscriber interface.

bobocep.cep.engine.decider.pubsub.BoboDeciderPublisher

class bobocep.cep.engine.decider.pubsub.**BoboDeciderPublisher**

Bases: ABC

A decider publisher interface.

__init__()

abstract snapshot() → Tuple[List[*BoboRunSerial*], List[*BoboRunSerial*], List[*BoboRunSerial*]]

A snapshot of the current state of the Decider.

Returns
Tuple of cached completed, cached halted, and currently partially-completed runs. If caching is disabled, the first two lists will be empty.

abstract subscribe(*subscriber:* BoboDeciderSubscriber)

Parameters
subscriber – Subscriber to add to list.

bobocep.cep.engine.decider.pubsub.BoboDeciderSubscriber

class bobocep.cep.engine.decider.pubsub.**BoboDeciderSubscriber**

Bases: ABC

A decider subscriber interface.

__init__()

abstract on_decider_update(*completed: List[*BoboRunSerial*], halted: List[*BoboRunSerial*], updated: List[*BoboRunSerial*], local: bool*) → None

Parameters

- **completed** – Completed runs.
- **halted** – Halted runs.
- **updated** – Updated runs.
- **local** – *True* if the Decider update occurred locally; *False* if the update occurred on a remote (distributed) instance.

bobocep.cep.engine.decider.run

A run.

Classes

BoboRun(run_id, phenomenon_name, pattern, ...)	A run that tracks the progress of a partially completed complex event.

bobocep.cep.engine.decider.run.BoboRun

class bobocep.cep.engine.decider.run.**BoboRun**(*run_id: str*, *phenomenon_name: str*, *pattern:* BoboPattern, *block_index: int*, *history:* BoboHistory)

Bases: `object`

A run that tracks the progress of a partially completed complex event.

__init__(*run_id: str*, *phenomenon_name: str*, *pattern:* BoboPattern, *block_index: int*, *history:* BoboHistory)

Parameters

- **run_id** – An ID for the run.
- **phenomenon_name** – A phenomenon name associated with the run.
- **pattern** – A pattern associated with the run.
- **block_index** – An index which indicates where in the pattern to start the run.
- **history** – A history of events for the run.

Raises

- ***BoboRunError*** – Run ID length is equal to 0.
- ***BoboRunError*** – Process name length is equal to 0.
- ***BoboRunError*** – Block index is less than 1.
- ***BoboRunError*** – History does not have enough events in it to cover all blocks up to the block index.

property block_index: int

Returns
The current block index of the run.

halt() → None

Halts the run.

history() → *BoboHistory*

Returns
The run history.

is_complete() → bool

Returns
True if run is completed; False otherwise.

is_halted() → bool

Returns
True if run has halted; False otherwise.

property pattern: ***BoboPattern***

Returns
The pattern associated with the run.

property phenomenon_name: str

Returns
The phenomenon name associated with the run.

process(*event:* BoboEvent) → bool

Parameters
event – An event for the run to process.

Returns
True if the event caused a state change in the run; *False* otherwise.

property run_id: str

Returns
The run ID.

serialize() → *BoboRunSerial*

Returns
A serializable representation of the run.

set_block(*block_index: int*, *history:* BoboHistory) → None

Updates the run's block to a new index and history.

Parameters

- **block_index** – The new block index.
- **history** – The new history.

Raises
BoboRunError: New block index is less than 1.

Raises

BoboRunError: New history does not have enough events in it to cover all blocks up to the new block index.

Exceptions

BoboRunError	A run error.

bobocep.cep.engine.decider.run.BoboRunError

exception bobocep.cep.engine.decider.run.**BoboRunError**

A run error.

bobocep.cep.engine.decider.runserial

A serializable representation of a run.

Classes

BoboRunSerial(run_id, phenomenon_name, ...)	Represents the state of a run and is designed to be serializable.

bobocep.cep.engine.decider.runserial.BoboRunSerial

class bobocep.cep.engine.decider.runserial.**BoboRunSerial**(*run_id: str*, *phenomenon_name: str*, *pattern_name: str*, *block_index: int*, *history:* BoboHistory)

Bases: *BoboJSONable*

Represents the state of a run and is designed to be serializable.

__init__(*run_id: str*, *phenomenon_name: str*, *pattern_name: str*, *block_index: int*, *history:* BoboHistory)

Parameters

- **run_id** – An ID for the run.
- **phenomenon_name** – A phenomenon name associated with the run.
- **pattern_name** – A pattern name associated with the run.
- **block_index** – An index which indicates where in the pattern to start the run.
- **history** – A history of events for the run.

Raises

- *BoboRunError* – Run ID length is equal to 0.
- *BoboRunError* – Process name length is equal to 0.
- *BoboRunError* – Block index is less than 1.

- *BoboRunError* – History does not have enough events in it to cover all blocks up to the block index.

property block_index: int

Returns
The current block index of the run.

static from_json_dict(*d: dict*) → *BoboRunSerial*

Parameters
d – A JSON *dict* representation of the event.

Returns
A new instance of the run serial.

static from_json_str(*j: str*) → *BoboRunSerial*

Parameters
j – A JSON *str* representation of the event.

Returns
A new instance of the run serial.

property history: *BoboHistory*

Returns
The run history.

property pattern_name: str

Returns
The pattern name associated with the run.

property phenomenon_name: str

Returns
The phenomenon name associated with the run.

property run_id: str

Returns
The run ID.

to_json_dict() → dict

Returns
A JSON *dict* representation of the run.

to_json_str() → str

Returns
A JSON *str* representation of the run.

Exceptions

BoboRunSerialError	A run serial error.

bobocep.cep.engine.decider.runserial.BoboRunSerialError

exception bobocep.cep.engine.decider.runserial.**BoboRunSerialError**

A run serial error.

bobocep.cep.engine.engine

CEP engine.

Classes

BoboEngine(receiver, decider, producer, ...)	The engine for complex event processing.

bobocep.cep.engine.engine.BoboEngine

class bobocep.cep.engine.engine.**BoboEngine**(*receiver: BoboReceiver*, *decider: BoboDecider*, *producer: BoboProducer*, *forwarder: BoboForwarder*, *times_receiver: int = 0*, *times_decider: int = 0*, *times_producer: int = 0*, *times_forwarder: int = 0*, *early_stop: bool = True*)

Bases: `object`

The engine for complex event processing.

__init__(*receiver: BoboReceiver*, *decider: BoboDecider*, *producer: BoboProducer*, *forwarder: BoboForwarder*, *times_receiver: int = 0*, *times_decider: int = 0*, *times_producer: int = 0*, *times_forwarder: int = 0*, *early_stop: bool = True*)

Parameters

- **receiver** – The receiver task.
- **decider** – The decider task.
- **producer** – The producer task.
- **forwarder** – The forwarder task.
- **times_receiver** – The number of times to run the receiver until moving to the decider task. A value of *0* runs the receiver indefinitely until it no longer performs an update of its internal state.
- **times_decider** – The number of times to run the decider until moving to the producer task. A value of *0* runs the decider indefinitely until it no longer performs an update of its internal state.
- **times_producer** – The number of times to run the producer until moving to the forwarder task. A value of *0* runs the producer indefinitely until it no longer performs an update of its internal state.

- **times_forwarder** – The number of times to run the forwarder until moving to the receiver task. A value of *0* runs the forwarder indefinitely until it no longer performs an update of its internal state.
- **early_stop** – If *times_** is greater than *0*, it will always run the task for the set number of times, even if the task does not update. Setting *early_stop* to *True* stops early if no task update occurs.

close() → None

Closes the engine.

property decider: *BoboDecider*

Get decider task.

property forwarder: *BoboForwarder*

Get forwarder task.

is_closed() → bool

Returns
True if engine is set to close; *False* otherwise.

property producer: *BoboProducer*

Get producer task.

property receiver: *BoboReceiver*

Get receiver task.

run() → None

Runs the engine. This is a blocking operation.

update() → bool

Updates the receiver, then the decider, then the producer, and finally to the forwarder. It updates each task *n* times, depending on how many times were chosen during engine instantiation.

Returns
True if engine is not set to close; *False* otherwise.

Exceptions

BoboEngineError	An engine error.

bobocep.cep.engine.engine.BoboEngineError

exception bobocep.cep.engine.engine.**BoboEngineError**

An engine error.

bobocep.cep.engine.forwarder

Forwarder imports.

Modules

bobocep.cep.engine.forwarder.forwarder	Engine task that forwards actions to external sources and generates action events.
bobocep.cep.engine.forwarder.pubsub	Forwarder publish-subscribe classes.

bobocep.cep.engine.forwarder.forwarder

Engine task that forwards actions to external sources and generates action events.

Classes

BoboForwarder(phenomena, handler, ...[, ...])	A forwarder task.

bobocep.cep.engine.forwarder.forwarder.BoboForwarder

class bobocep.cep.engine.forwarder.forwarder.**BoboForwarder**(*phenomena: List[*BoboPhenomenon*]*, *handler:* BoboActionHandler, *gen_event_id:* BoboGenEventID, *gen_timestamp:* BoboGenTimestamp, *local_only: bool = True*, *max_size: int = 0*)

Bases: *BoboEngineTask*, *BoboForwarderPublisher*, *BoboProducerSubscriber*

A forwarder task.

__init__(*phenomena: List[*BoboPhenomenon*]*, *handler:* BoboActionHandler, *gen_event_id:* BoboGenEventID, *gen_timestamp:* BoboGenTimestamp, *local_only: bool = True*, *max_size: int = 0*)

Parameters

- **phenomena** – List of phenomena.
- **handler** – Action handler.
- **gen_event_id** – Event ID generator.
- **gen_timestamp** – Timestamp generator.
- **local_only** – If *True*, forwarder only executes actions for locally-completed complex events i.e. not complex events that were completed on a distributed instance. **Note**: if *False*, the action may be executed more than once: once by the instance that generated the remote event, and another by this instance.
- **max_size** – Maximum queue size. Default: 0 (unbounded).

`close()` → None

Closes the Forwarder.

`is_closed()` → bool

Returns
True if Forwarder is closed; *False* otherwise.

`on_producer_update`(*event: BoboEventComplex*, *local: bool*) → None

Parameters

- **`event`** – Complex event generated by Producer.
- **`local`** – *True* if the complex event was generated using a locally-completed run; *False* otherwise.

`size()` → int

Returns
Queue size.

`subscribe`(*subscriber: BoboForwarderSubscriber*)

Parameters
`subscriber` – Subscriber to Forwarder data.

`update()` → bool

Returns
True if an internal update occurred; *False* otherwise.

Exceptions

BoboForwarderError	A forwarder task error.

bobocep.cep.engine.forwarder.forwarder.BoboForwarderError

exception `bobocep.cep.engine.forwarder.forwarder.`**`BoboForwarderError`**

A forwarder task error.

bobocep.cep.engine.forwarder.pubsub

Forwarder publish-subscribe classes.

Classes

BoboForwarderPublisher()	A forwarder publisher interface.
BoboForwarderSubscriber()	A forwarder subscriber interface.

bobocep.cep.engine.forwarder.pubsub.BoboForwarderPublisher

class bobocep.cep.engine.forwarder.pubsub.**BoboForwarderPublisher**

Bases: ABC

A forwarder publisher interface.

__init__()

abstract subscribe(*subscriber:* BoboForwarderSubscriber)

Parameters

subscriber – Subscriber to add to list.

bobocep.cep.engine.forwarder.pubsub.BoboForwarderSubscriber

class bobocep.cep.engine.forwarder.pubsub.**BoboForwarderSubscriber**

Bases: ABC

A forwarder subscriber interface.

__init__()

abstract on_forwarder_update(*event:* BoboEventAction) → None

Parameters

event – Action event generated by Forwarder.

bobocep.cep.engine.producer

Producer imports.

Modules

bobocep.cep.engine.producer.producer	Engine task that generates complex events and triggers actions.
bobocep.cep.engine.producer.pubsub	Producer publish-subscribe classes.

bobocep.cep.engine.producer.producer

Engine task that generates complex events and triggers actions.

Classes

BoboProducer(phenomena, gen_event_id, ...[, ...])	A producer task.

bobocep.cep.engine.producer.producer.BoboProducer

class bobocep.cep.engine.producer.producer.**BoboProducer**(*phenomena: List[*BoboPhenomenon*]*, *gen_event_id:* BoboGenEventID, *gen_timestamp:* BoboGenTimestamp, *max_size: int = 0*)

Bases: *BoboEngineTask*, *BoboProducerPublisher*, *BoboDeciderSubscriber*

A producer task.

__init__(*phenomena: List[*BoboPhenomenon*]*, *gen_event_id:* BoboGenEventID, *gen_timestamp:* BoboGenTimestamp, *max_size: int = 0*)

Parameters

- **phenomena** – List of phenomena.
- **gen_event_id** – Event ID generator.
- **gen_timestamp** – Timestamp generator.
- **max_size** – Maximum queue size. Default: 0 (unbounded).

close() → None

Closes the Producer.

is_closed() → bool

Returns

True if Producer is closed; *False* otherwise.

on_decider_update(*completed: List[*BoboRunSerial*]*, *halted: List[*BoboRunSerial*]*, *updated: List[*BoboRunSerial*]*, *local: bool*) → None

Parameters

- **completed** – Completed runs.
- **halted** – Halted runs.
- **updated** – Updated runs.
- **local** – *True* if the Decider update occurred locally; *False* if the update occurred on a remote (distributed) instance.

size() → int

Returns

Queue size.

subscribe(*subscriber:* BoboProducerSubscriber)

Parameters
subscriber – Subscriber to Producer data.

update() → bool

Returns
True if an internal update occurred; *False* otherwise.

Exceptions

BoboProducerError	A producer task error.

bobocep.cep.engine.producer.producer.BoboProducerError

exception bobocep.cep.engine.producer.producer.**BoboProducerError**

A producer task error.

bobocep.cep.engine.producer.pubsub

Producer publish-subscribe classes.

Classes

BoboProducerPublisher()	A producer publisher interface.
BoboProducerSubscriber()	A producer subscriber interface.

bobocep.cep.engine.producer.pubsub.BoboProducerPublisher

class bobocep.cep.engine.producer.pubsub.**BoboProducerPublisher**

Bases: ABC

A producer publisher interface.

__init__()

abstract subscribe(*subscriber:* BoboProducerSubscriber)

Parameters
subscriber – Subscriber to add to list.

bobocep.cep.engine.producer.pubsub.BoboProducerSubscriber

class bobocep.cep.engine.producer.pubsub.**BoboProducerSubscriber**

Bases: ABC

A producer subscriber interface.

__init__()

abstract on_producer_update(*event: BoboEventComplex*, *local: bool*) → None

Parameters

- **event** – Complex event generated by Producer.
- **local** – *True* if the complex event was generated using a locally-completed run; *False* otherwise.

bobocep.cep.engine.receiver

Receiver imports.

Modules

bobocep.cep.engine.receiver.pubsub	Receiver publish-subscribe classes.
bobocep.cep.engine.receiver.receiver	Engine task that provides an entry point for data into the system.
bobocep.cep.engine.receiver.validator	Receiver data validators.

bobocep.cep.engine.receiver.pubsub

Receiver publish-subscribe classes.

Classes

BoboReceiverPublisher()	A receiver publisher interface.
BoboReceiverSubscriber()	A receiver subscriber interface.

bobocep.cep.engine.receiver.pubsub.BoboReceiverPublisher

class bobocep.cep.engine.receiver.pubsub.**BoboReceiverPublisher**

Bases: ABC

A receiver publisher interface.

__init__()

abstract subscribe(*subscriber: BoboReceiverSubscriber*) → None

Parameters
subscriber – Subscriber to add to list.

bobocep.cep.engine.receiver.pubsub.BoboReceiverSubscriber

class bobocep.cep.engine.receiver.pubsub.**BoboReceiverSubscriber**

Bases: ABC

A receiver subscriber interface.

__init__()

abstract on_receiver_update(*event: BoboEvent*) → None

Parameters
event – A new BoboEvent instance processed by the receiver.

bobocep.cep.engine.receiver.receiver

Engine task that provides an entry point for data into the system.

Classes

BoboReceiver(validator, gen_event_id, ...[, ...])	A receiver task.

bobocep.cep.engine.receiver.receiver.BoboReceiver

class bobocep.cep.engine.receiver.receiver.**BoboReceiver**(*validator: BoboValidator, gen_event_id: BoboGenEventID, gen_timestamp: BoboGenTimestamp, gen_event: BoboGenEvent | None = None, max_size: int = 0*)

Bases: *BoboEngineTask*, *BoboReceiverPublisher*, *BoboProducerSubscriber*, *BoboForwarderSubscriber*

A receiver task.

__init__(*validator: BoboValidator, gen_event_id: BoboGenEventID, gen_timestamp: BoboGenTimestamp, gen_event: BoboGenEvent | None = None, max_size: int = 0*)

Parameters

- **validator** – Incoming data validator.
- **gen_event_id** – Event ID generator.
- **gen_timestamp** – Timestamp generator.
- **gen_event** – Event generator (optional).
- **max_size** – Maximum queue size. Default: 0 (unbounded).

`add_data`(*data: Any*) → None

Parameters
`data` – Data to add to the receiver.

Raises
BoboReceiverError – If receiver queue is full.

`close`() → None

Closes the Receiver.

`is_closed`() → bool

Returns
True if Receiver is closed; *False* otherwise.

`on_forwarder_update`(*event:* BoboEventAction) → None

Parameters
`event` – Action event generated by Forwarder.

`on_producer_update`(*event:* BoboEventComplex, *local: bool*) → None

Parameters

- `event` – Complex event generated by Producer.
- `local` – *True* if the complex event was generated using a locally-completed run; *False* otherwise.

`size`() → int

Returns
Queue size.

`subscribe`(*subscriber:* BoboReceiverSubscriber) → None

Parameters
`subscriber` – Subscriber to Receiver data.

`update`() → bool

Processes data from its queue, if any. Also processes a generated event if receiver is set to generate any. A BoboEventSimple instance is produced for data if they pass validation and are not already BoboEvent instances. Valid data are then sent to receiver subscribers.

Returns
True if queue or generated data are processed; *False* otherwise.

Exceptions

BoboReceiverError	A receiver task error.

bobocep.cep.engine.receiver.receiver.BoboReceiverError

exception bobocep.cep.engine.receiver.receiver.**BoboReceiverError**

A receiver task error.

bobocep.cep.engine.receiver.validator

Receiver data validators.

Classes

BoboValidator()	An abstract validator.
BoboValidatorAll()	Validator that accepts all data.
BoboValidatorJSONSchema(schema)	Validates whether the data type is valid with respect to a given JSON Schema.
BoboValidatorJSONable()	Validates whether the data type is JSONable.
BoboValidatorType(types[, subtype])	Validates whether the entity type matches any of the given data types.

bobocep.cep.engine.receiver.validator.BoboValidator

class bobocep.cep.engine.receiver.validator.**BoboValidator**

Bases: ABC

An abstract validator.

__init__()

abstract is_valid(*data: Any*) → bool

Parameters
data – Data to validate.

Returns
True if data are valid; *False* otherwise.

bobocep.cep.engine.receiver.validator.BoboValidatorAll

class bobocep.cep.engine.receiver.validator.**BoboValidatorAll**

Bases: *BoboValidator*

Validator that accepts all data.

__init__()

is_valid(*data: Any*) → bool

Returns
Always returns *True*.

bobocep.cep.engine.receiver.validator.BoboValidatorJSONSchema

class bobocep.cep.engine.receiver.validator.**BoboValidatorJSONSchema**(*schema: dict*)

Bases: *BoboValidatorJSONable*

Validates whether the data type is valid with respect to a given JSON Schema. If the data are a BoboEvent, then the event's data are checked instead.

__init__(*schema: dict*)

Parameters

schema – The JSON schema against which to compare data.

is_valid(*data: Any*) → bool

Returns

True if data are valid as per the JSON schema; *False* otherwise.

Raises

BoboValidatorError: Invalid JSON schema.

bobocep.cep.engine.receiver.validator.BoboValidatorJSONable

class bobocep.cep.engine.receiver.validator.**BoboValidatorJSONable**

Bases: *BoboValidator*

Validates whether the data type is JSONable. If the data are a BoboEvent, then the event's data are checked instead.

__init__()

is_valid(*data: Any*) → bool

Returns

True if data are valid JSON; *False* otherwise.

bobocep.cep.engine.receiver.validator.BoboValidatorType

class bobocep.cep.engine.receiver.validator.**BoboValidatorType**(*types: List[type]*, *subtype: bool = True*)

Bases: *BoboValidator*

Validates whether the entity type matches any of the given data types. If the data are a BoboEvent, then the event's data are checked instead.

__init__(*types: List[type]*, *subtype: bool = True*)

Parameters

- **types** – The types of which the data must match at least one for the data to be valid.
- **subtype** – If *True*, subtypes of a type are also valid. If *False*, data must match exact type.

is_valid(*data: Any*) → bool

Returns

True if the data matches a type; *False* otherwise.

Exceptions

BoboValidatorError	A validator error.

bobocep.cep.engine.receiver.validator.BoboValidatorError

exception bobocep.cep.engine.receiver.validator.**BoboValidatorError**

A validator error.

bobocep.cep.engine.task

CEP engine tasks.

Classes

BoboEngineTask()	An engine task.

bobocep.cep.engine.task.BoboEngineTask

class bobocep.cep.engine.task.**BoboEngineTask**

Bases: ABC

An engine task.

__init__()

abstract close() → None

Closes the engine task.

abstract is_closed() → bool

Returns
True if task is set to close; *False* otherwise.

abstract size() → int

Returns
The number of events that the task is currently handling.

abstract update() → bool

Returns
True if an update occurred in the task; *False* otherwise.

Exceptions

BoboEngineTaskError	An engine task error.

bobocep.cep.engine.task.BoboEngineTaskError

exception bobocep.cep.engine.task.**BoboEngineTaskError**

An engine task error.

bobocep.cep.event

Event imports.

Modules

bobocep.cep.event.action	Action event.
bobocep.cep.event.complex	Complex event.
bobocep.cep.event.event	Abstract event.
bobocep.cep.event.factory	Event factory.
bobocep.cep.event.history	Event history.
bobocep.cep.event.simple	Simple event.

bobocep.cep.event.action

Action event.

Classes

BoboEventAction(event_id, timestamp, data, ...)	An action event.

bobocep.cep.event.action.BoboEventAction

class bobocep.cep.event.action.**BoboEventAction**(*event_id*, *timestamp: int*, *data: Any*, *phenomenon_name: str*, *pattern_name: str*, *action_name: str*, *success: bool*)

Bases: *BoboEvent*

An action event.

__init__(*event_id*, *timestamp: int*, *data: Any*, *phenomenon_name: str*, *pattern_name: str*, *action_name: str*, *success: bool*)

Parameters

- **event_id** – The event ID.

- `timestamp` – The event timestamp.
- `data` – The event data.
- `phenomenon_name` – The phenomenon name.
- `pattern_name` – The pattern name.
- `action_name` – The action name.
- `success` – *True* if the action was successful; *False* otherwise.

Raises

- *BoboEventError* – If length of phenomenon name is equal to 0.
- *BoboEventError* – If length of pattern name is equal to 0.
- *BoboEventError* – If length of action name is equal to 0.

property action_name: str

Returns
Action name.

cast(*dtype: type*) → *BoboEventAction*

Parameters
`dtype` – The type to which the event's data is cast.

Returns
A new BoboEventAction instance with its data cast to *dtype* and all other properties identical to the original event.

property data: Any

Get event data.

property event_id: str

Get event ID.

static from_json_dict(*d: dict*) → *BoboEventAction*

Parameters
`d` – A JSON *dict* representation of the event.

Returns
A new instance of the event type.

static from_json_str(*j: str*) → *BoboEventAction*

Parameters
`j` – A JSON *str* representation of the event.

Returns
A new instance of the event type.

property pattern_name: str

Returns
Pattern name.

property phenomenon_name: str

Returns
Phenomenon name.

property success: bool

Returns
True if action was executed successfully; *False* otherwise.

property timestamp: int

Get event timestamp.

to_json_dict() → dict

Returns
A JSON *dict* representation of the event.

to_json_str() → str

Returns
A JSON *str* representation of the event.

bobocep.cep.event.complex

Complex event.

Classes

BoboEventComplex(event_id, timestamp, data, ...)	A complex event.

bobocep.cep.event.complex.BoboEventComplex

class bobocep.cep.event.complex.**BoboEventComplex**(*event_id: str*, *timestamp: int*, *data: Any*, *phenomenon_name: str*, *pattern_name: str*, *history:* BoboHistory)

Bases: *BoboEvent*

A complex event.

__init__(*event_id: str*, *timestamp: int*, *data: Any*, *phenomenon_name: str*, *pattern_name: str*, *history:* BoboHistory)

Parameters

- **event_id** – The event ID.
- **timestamp** – The event timestamp.
- **data** – The event data.
- **phenomenon_name** – The phenomenon name.
- **pattern_name** – The pattern name.
- **history** – The history of events.

Raises

- *BoboEventError* – If length of phenomenon name is equal to 0.
- *BoboEventError* – If length of pattern name is equal to 0.

`cast`(*dtype: type*) → *BoboEventComplex*

Parameters
`dtype` – The type to which the event's data is cast.

Returns
A new BoboEventComplex instance with its data cast to *dtype* and all other properties identical to the original event.

property data: Any

Get event data.

property event_id: str

Get event ID.

static from_json_dict(*d: dict*) → *BoboEventComplex*

Parameters
`d` – A JSON *dict* representation of the event.

Returns
A new instance of the event type.

static from_json_str(*j: str*) → *BoboEventComplex*

Parameters
`j` – A JSON *str* representation of the event.

Returns
A new instance of the event type.

property history: ***BoboHistory***

Returns
Event history.

property pattern_name: str

Returns
Pattern name.

property phenomenon_name: str

Returns
Phenomenon name.

property timestamp: int

Get event timestamp.

`to_json_dict`() → dict

Returns
A JSON *dict* representation of the event.

`to_json_str`() → str

Returns
A JSON *str* representation of the event.

bobocep.cep.event.event

Abstract event.

Classes

BoboEvent(event_id, timestamp, data)	An abstract event.

bobocep.cep.event.event.BoboEvent

class bobocep.cep.event.event.**BoboEvent**(*event_id: str*, *timestamp: int*, *data: Any*)

Bases: *BoboJSONable*, ABC

An abstract event.

__init__(*event_id: str*, *timestamp: int*, *data: Any*)

Parameters

- **event_id** – The event ID.
- **timestamp** – The event timestamp.
- **data** – The event data.

Raises

BoboEventError – If length of event ID is equal to 0.

abstract cast(*dtype: type*) → *BoboEvent*

Parameters

dtype – The type to which the event's data is cast.

Returns

A new BoboEvent instance with its data cast to *dtype* and all other properties identical to the original event.

property data: Any

Get event data.

property event_id: str

Get event ID.

abstract static from_json_dict(*d: dict*) → *BoboJSONable*

Parameters

d – A JSON *dict* representation of an object of this type.

Returns

A new instance of this type.

abstract static from_json_str(*j: str*) → *BoboJSONable*

Parameters

j – A JSON *str* representation of an object of this type.

Returns

A new instance of this type.

property timestamp: int

Get event timestamp.

abstract to_json_dict() → dict

Returns
A JSON *dict* representation of an object of this type.

abstract to_json_str() → str

Returns
A JSON *str* representation of an object of this type.

Exceptions

BoboEventError	An event error.

bobocep.cep.event.event.BoboEventError

exception bobocep.cep.event.event.**BoboEventError**

An event error.

bobocep.cep.event.factory

Event factory.

Classes

BoboEventFactory()	A BoboEvent factory that generates instances from JSON representations of events.

bobocep.cep.event.factory.BoboEventFactory

class bobocep.cep.event.factory.**BoboEventFactory**

Bases: object

A BoboEvent factory that generates instances from JSON representations of events.

__init__()

static from_json_str(*j: str*) → *BoboEvent*

Parameters
j – A JSON *str* representation of the event.

Returns
A new instance of the event type.

Raises

- *BoboEventFactoryError* – If *EVENT_TYPE* key is missing from JSON.
- *BoboEventFactoryError* – If *EVENT_TYPE* value is an unknown event type.

Exceptions

BoboEventFactoryError	An event factory error.

bobocep.cep.event.factory.BoboEventFactoryError

exception bobocep.cep.event.factory.**BoboEventFactoryError**

An event factory error.

bobocep.cep.event.history

Event history.

Classes

BoboHistory(events)	An event history.

bobocep.cep.event.history.BoboHistory

class bobocep.cep.event.history.**BoboHistory**(*events: Dict[str, List[BoboEvent]]*)

Bases: *BoboJSONable*

An event history.

__init__(*events: Dict[str, List[BoboEvent]]*)

Parameters

events – The history of events. Keys are group names. Values are lists of BoboEvent instances associated with a group.

all_events() → Tuple[*BoboEvent*, ...]

Returns

All history events in a tuple.

all_groups() → Tuple[str, ...]

Returns

All history groups in a tuple.

property events: Dict[str, List[*BoboEvent*]]

Returns

All history events, indexed by group.

`first()` → *BoboEvent* | None

Returns
The BoboEvent with the oldest timestamp, if there is at least one BoboEvent in the history.

`static from_json_dict(d: dict)` → *BoboHistory*

Parameters
`d` – A JSON *dict* representation of the history.

Returns
A new instance of the history.

`static from_json_str(j: str)` → *BoboHistory*

Parameters
`j` – A JSON *str* representation of the history.

Returns
A new instance of the history.

`group(group: str)` → Tuple[*BoboEvent*, ...]

Parameters
`group` – A group name.

Returns
The BoboEvent instances associated with *group*.

`last()` → *BoboEvent* | None

Returns
The BoboEvent with the most recent timestamp, if there is at least one BoboEvent in the history.

`size()` → int

Returns
The total number of history events across all groups.

`to_json_dict()` → dict

Returns
A JSON *dict* representation of the history.

`to_json_str()` → str

Returns
A JSON *str* representation of the history.

bobocep.cep.event.simple

Simple event.

Classes

`BoboEventSimple`(event_id, timestamp, data)	A simple event.

bobocep.cep.event.simple.BoboEventSimple

class `bobocep.cep.event.simple.BoboEventSimple`(*event_id: str*, *timestamp: int*, *data: Any*)

Bases: `BoboEvent`

A simple event.

`__init__`(*event_id: str*, *timestamp: int*, *data: Any*)

Parameters

- **event_id** – The event ID.
- **timestamp** – The event timestamp.
- **data** – The event data.

`cast`(*dtype: type*) → *BoboEventSimple*

Parameters
dtype – The type to which the event's data is cast.

Returns
A new BoboEventSimple instance with its data cast to *dtype* and all other properties identical to the original event.

property data: Any

Get event data.

property event_id: str

Get event ID.

static from_json_dict(*d: dict*) → *BoboEventSimple*

Parameters
d – A JSON *dict* representation of the event.

Returns
A new instance of the event type.

static from_json_str(*j: str*) → *BoboEventSimple*

Parameters
j – A JSON *str* representation of the event.

Returns
A new instance of the event type.

property timestamp: int

Get event timestamp.

`to_json_dict`() → dict

Returns
A JSON *dict* representation of the event.

to_json_str() → str

Returns
A JSON *str* representation of the event.

bobocep.cep.gen

Generator imports.

Modules

bobocep.cep.gen.event	Generates BoboEvent objects.
bobocep.cep.gen.event_id	Generates BoboEvent IDs.
bobocep.cep.gen.timestamp	Generates BoboEvent timestamps.

bobocep.cep.gen.event

Generates BoboEvent objects.

Classes

BoboGenEvent()	An event generator.
BoboGenEventNone()	An event generator that always returns None.
BoboGenEventTime(millis[, datagen, ...])	An event generator that returns a BoboEventSimple time event if a given amount of time has elapsed.

bobocep.cep.gen.event.BoboGenEvent

class bobocep.cep.gen.event.**BoboGenEvent**

Bases: ABC

An event generator.

__init__()

abstract **maybe_generate**(*event_id: str*) → *BoboEvent* | None

Parameters
event_id – The event ID to use in the generated event.

Returns
Either a generated BoboEvent instance or None.

bobocep.cep.gen.event.BoboGenEventNone

class bobocep.cep.gen.event.**BoboGenEventNone**

Bases: *BoboGenEvent*

An event generator that always returns None.

__init__()

maybe_generate(*event_id: str*) → *BoboEvent* | None

Parameters
event_id – An event ID (always ignored).

Returns
None.

bobocep.cep.gen.event.BoboGenEventTime

class bobocep.cep.gen.event.**BoboGenEventTime**(*millis: int*, *datagen: Callable* | *None = None*, *gen_timestamp:* BoboGenTimestamp | *None = None*, *from_now: bool = True*, *tz=None*)

Bases: *BoboGenEvent*

An event generator that returns a BoboEventSimple time event if a given amount of time has elapsed. If the time has not elapsed, None is returned instead.

__init__(*millis: int*, *datagen: Callable* | *None = None*, *gen_timestamp:* BoboGenTimestamp | *None = None*, *from_now: bool = True*, *tz=None*)

Parameters

- **millis** – Milliseconds between event generations. Once the millisecond timer has elapsed, an event can be generated.
- **datagen** – Datagen to use in generated event.
- **gen_timestamp** – Custom timestamp generator. If None, a BoboGenTimestampEpoch instance is used to generate timestamps.
- **from_now** – If *True*, sets the millisecond timer to the current time; otherwise, the timer is set to 0.
- **tz** – Timezone data.

maybe_generate(*event_id: str*) → *BoboEvent* | None

Parameters
event_id – An event ID.

Returns
A BoboEventSimple instance if one is due to be generated, or *None* if not.

bobocep.cep.gen.event_id

Generates BoboEvent IDs.

Classes

BoboGenEventID()	An event ID generator.
BoboGenEventIDUnique([urn])	An event ID generator that always generates a unique, non-repeating ID.

bobocep.cep.gen.event_id.BoboGenEventID

`class bobocep.cep.gen.event_id.BoboGenEventID`

Bases: `ABC`

An event ID generator.

`__init__()`

`abstract generate()` → str

Returns
Generated event ID.

bobocep.cep.gen.event_id.BoboGenEventIDUnique

`class bobocep.cep.gen.event_id.BoboGenEventIDUnique`(*urn: str | None = None*)

Bases: *BoboGenEventID*

An event ID generator that always generates a unique, non-repeating ID.

`__init__`(*urn: str | None = None*)

Parameters
urn – A URN to prefix before the generated event ID (optional).

`generate()` → str

Returns
A generated event ID.

bobocep.cep.gen.timestamp

Generates BoboEvent timestamps.

Classes

BoboGenTimestamp()	A timestamp generator.
BoboGenTimestampEpoch()	A timestamp generator that returns the current time, in milliseconds, since the Epoch.

bobocep.cep.gen.timestamp.BoboGenTimestamp

class bobocep.cep.gen.timestamp.**BoboGenTimestamp**

Bases: ABC

A timestamp generator.

__init__()

abstract generate() → int

Returns
Generated timestamp.

bobocep.cep.gen.timestamp.BoboGenTimestampEpoch

class bobocep.cep.gen.timestamp.**BoboGenTimestampEpoch**

Bases: *BoboGenTimestamp*

A timestamp generator that returns the current time, in milliseconds, since the Epoch.

__init__()

generate() → int

Returns
The current time, in milliseconds, since the Epoch.

bobocep.cep.phenom

Phenomenon imports.

Modules

bobocep.cep.phenom.pattern	Pattern imports.
bobocep.cep.phenom.phenom	A phenomenon.

bobocep.cep.phenom.pattern

Pattern imports.

Modules

bobocep.cep.phenom.pattern.builder	Builders to assist in generating a pattern.
bobocep.cep.phenom.pattern.pattern	A pattern which models the occurrence of a phenomenon and facilitates the generating of a complex event.
bobocep.cep.phenom.pattern.predicate	A predicate tested against an event that must evaluate to *True* for the event to be accepted as being part of the manifestation of a complex event.

bobocep.cep.phenom.pattern.builder

Builders to assist in generating a pattern.

Classes

BoboPatternBuilder(name[, singleton])	A pattern builder.

bobocep.cep.phenom.pattern.builder.BoboPatternBuilder

class bobocep.cep.phenom.pattern.builder.**BoboPatternBuilder**(*name: str*, *singleton: bool = False*)

Bases: `object`

A pattern builder.

__init__(*name: str*, *singleton: bool = False*)

Parameters

name – Pattern name.

followed_by(*predicate:* BoboPredicate | *Callable*, *group: str = ''*, *times: int = 1*, *loop: bool = False*, *optional: bool = False*) → *BoboPatternBuilder*

Adds a block with relaxed contiguity.

Parameters

- **predicate** – Block predicate. If a Callable is provided, it will be wrapped in a BoboPredicateCall instance.
- **group** – Block group.
- **times** – Number of times to add this block to the pattern, in sequence.
- **loop** – If *True*, the block loops back onto itself.
- **optional** – If *True*, the block is optional.

Returns

The BoboPatternBuilder instance that made the function call.

followed_by_any(*predicates: List[*BoboPredicate *| Callable], group: str = '', times: int = 1, loop: bool = False, optional: bool = False*) → *BoboPatternBuilder*

Adds multiple blocks with non-deterministic relaxed contiguity.

Parameters

- **predicates** – Predicates, one per block. Any Callable types in the list will be wrapped in their own BoboPredicateCall instance.
- **group** – Group name for all blocks.
- **times** – Number of times to add these blocks to the pattern, in sequence.
- **loop** – If *True*, the blocks loop back onto themselves.
- **optional** – If *True*, the blocks are optional.

Returns

The BoboPatternBuilder instance that made the function call.

generate() → *BoboPattern*

Generates a BoboPattern instance with the configuration specified in the builder.

Returns

A BoboPattern instance.

haltcondition(*predicate:* BoboPredicate | *Callable*) → *BoboPatternBuilder*

Adds a haltcondition.

Parameters

predicate – The haltcondition predicate. If a Callable is provided, it will be wrapped in a BoboPredicateCall instance.

Returns

The BoboPatternBuilder instance that made the function call.

next(*predicate:* BoboPredicate | *Callable, group: str = '', times: int = 1, loop: bool = False*) → *BoboPatternBuilder*

Adds a block with strict contiguity.

Parameters

- **predicate** – Block predicate. If a Callable is provided, it will be wrapped in a BoboPredicateCall instance.
- **group** – Block group.
- **times** – Number of times to add this block to the pattern, in sequence.
- **loop** – If *True*, the block loops back onto itself (making it non-deterministic). If *False*, the block remains deterministic.

Returns

The BoboPatternBuilder instance that made the function call.

not_followed_by(*predicate:* BoboPredicate | *Callable, group: str = '', times: int = 1*) → *BoboPatternBuilder*

Adds a negated block with relaxed contiguity.

Parameters

- **predicate** – Block predicate. If a Callable is provided, it will be wrapped in a BoboPredicateCall instance.

- **group** – Block group.
- **times** – Number of times to add this block to the pattern, in sequence.

Returns
The BoboPatternBuilder instance that made the function call.

not_followed_by_any(*predicates: List[BoboPredicate | Callable], group: str = '', times: int = 1*) → *BoboPatternBuilder*

Adds multiple negated blocks with non-deterministic relaxed contiguity.

Parameters

- **predicates** – Predicates, one per block. Any Callable types in the list will be wrapped in their own BoboPredicateCall instance.
- **group** – Group name for all blocks.
- **times** – Number of times to add these blocks to the pattern, in sequence.

Returns
The BoboPatternBuilder instance that made the function call.

not_next(*predicate: BoboPredicate | Callable, group: str = '', times: int = 1*) → *BoboPatternBuilder*

Adds a negated block with strict contiguity.

Parameters

- **predicate** – Block predicate. If a Callable is provided, it will be wrapped in a BoboPredicateCall instance.
- **group** – Block group.
- **times** – Number of times to add this block to the pattern, in sequence.

Returns
The BoboPatternBuilder instance that made the function call.

precondition(*predicate: BoboPredicate | Callable*) → *BoboPatternBuilder*

Adds a precondition.

Parameters
predicate – The precondition predicate. If a Callable is provided, it will be wrapped in a BoboPredicateCall instance.

Returns
The BoboPatternBuilder instance that made the function call.

Exceptions

BoboPatternBuilderError	A pattern builder error.

bobocep.cep.phenom.pattern.builder.BoboPatternBuilderError

exception bobocep.cep.phenom.pattern.builder.**BoboPatternBuilderError**

A pattern builder error.

bobocep.cep.phenom.pattern.pattern

A pattern which models the occurrence of a phenomenon and facilitates the generating of a complex event.

Classes

BoboPattern(name, blocks, preconditions, ...)	A pattern that represents a means by which to detect the occurrence of some phenomenon.
BoboPatternBlock(predicates, group, strict, ...)	A pattern block.

bobocep.cep.phenom.pattern.pattern.BoboPattern

class bobocep.cep.phenom.pattern.pattern.**BoboPattern**(*name: str*, *blocks: List[*BoboPatternBlock*]*, *preconditions: List[*BoboPredicate*]*, *haltconditions: List[*BoboPredicate*]*, *singleton: bool = False*)

Bases: object

A pattern that represents a means by which to detect the occurrence of some phenomenon.

__init__(*name: str*, *blocks: List[*BoboPatternBlock*]*, *preconditions: List[*BoboPredicate*]*, *haltconditions: List[*BoboPredicate*]*, *singleton: bool = False*)

Parameters

- **name** – The pattern name.
- **blocks** – The pattern blocks.
- **preconditions** – The pattern preconditions.
- **haltconditions** – The pattern haltconditions.
- **singleton** – If *True*, the pattern can only have one active run at a time.

property blocks: Tuple[*BoboPatternBlock*, ...]

Returns
Pattern blocks.

property haltconditions: Tuple[*BoboPredicate*, ...]

Returns
Pattern haltconditions.

property name: str

Returns
Pattern name.

property preconditions: Tuple[*BoboPredicate*, ...]

Returns
Pattern preconditions.

property singleton: bool

Returns
True if pattern can only have one active run at a time; *False* otherwise.

bobocep.cep.phenom.pattern.pattern.BoboPatternBlock

class bobocep.cep.phenom.pattern.pattern.**BoboPatternBlock**(*predicates: List[*BoboPredicate*]*, *group: str*, *strict: bool*, *loop: bool*, *negated: bool*, *optional: bool*)

Bases: object

A pattern block.

__init__(*predicates: List[*BoboPredicate*]*, *group: str*, *strict: bool*, *loop: bool*, *negated: bool*, *optional: bool*)

Parameters

- **predicates** – Block predicates.
- **group** – The group with which the block is associated. Can be an empty string.
- **strict** – *True* if the block has strict contiguity; *False* otherwise.
- **loop** – *True* if the block loops back to itself; *False* otherwise.
- **negated** – *True* if the block is negated; *False* otherwise.
- **optional** – *True* if the block is optional; *False* otherwise.

property group: str

Returns
Block group.

property loop: bool

Returns
True if pattern block loops; *False* otherwise.

property negated: bool

Returns
True if pattern block is negated; *False* otherwise.

property optional: bool

Returns
True if pattern block is optional; *False* otherwise.

property predicates: Tuple[*BoboPredicate*, ...]

Returns
Block predicates.

property strict: bool

Returns
True if pattern block has strict contiguity; *False* otherwise.

Exceptions

BoboPatternBlockError	A pattern block error.
BoboPatternError	A pattern error.

bobocep.cep.phenom.pattern.pattern.BoboPatternBlockError

exception bobocep.cep.phenom.pattern.pattern.**BoboPatternBlockError**

A pattern block error.

bobocep.cep.phenom.pattern.pattern.BoboPatternError

exception bobocep.cep.phenom.pattern.pattern.**BoboPatternError**

A pattern error.

bobocep.cep.phenom.pattern.predicate

A predicate tested against an event that must evaluate to *True* for the event to be accepted as being part of the manifestation of a complex event.

Classes

BoboPredicate()	A predicate that evaluates to either True or False.
BoboPredicateCall(call)	A predicate that evaluates using a custom function or method (i.e. a 'callable').
BoboPredicateCallType(call, dtype[, ...])	A predicate that evaluates using a custom function or method after first checking whether the event data is an instance of a given type.

bobocep.cep.phenom.pattern.predicate.BoboPredicate

class bobocep.cep.phenom.pattern.predicate.**BoboPredicate**

Bases: ABC

A predicate that evaluates to either True or False.

__init__()

abstract evaluate(*event: BoboEvent*, *history: BoboHistory*) → bool

Evaluates the predicate.

Parameters

- **event** (BoboEvent) – An event.
- **history** (BoboHistory) – A history of events.

Returns
True if the predicate evaluates to True, False otherwise.

Return type
bool

bobocep.cep.phenom.pattern.predicate.BoboPredicateCall

class bobocep.cep.phenom.pattern.predicate.**BoboPredicateCall**(*call: Callable*)

Bases: *BoboPredicate*

A predicate that evaluates using a custom function or method (i.e. a 'callable').

__init__(*call: Callable*)

Parameters
call – The callable to use for evaluating the predicate.

evaluate(*event: BoboEvent*, *history: BoboHistory*) → bool

Parameters

- **event** – The event used for evaluation.
- **history** – The history of currently accepted events.

Returns
True if predicate is satisfied; *False* otherwise.

bobocep.cep.phenom.pattern.predicate.BoboPredicateCallType

class bobocep.cep.phenom.pattern.predicate.**BoboPredicateCallType**(*call: Callable*, *dtype: type*, *subtype: bool = True*, *cast: bool = True*)

Bases: *BoboPredicateCall*

A predicate that evaluates using a custom function or method after first checking whether the event data is an instance of a given type. If it is not, then a cast to the type can be attempted and a **copy of the event** is passed with its data cast to the type.

Note: the copy is only used **within the predicate callable**. The original event remains in use elsewhere.

__init__(*call: Callable*, *dtype: type*, *subtype: bool = True*, *cast: bool = True*)

Parameters

- **call** – The callable to use for evaluating the predicate.
- **dtype** – The data type to use for evaluation.
- **subtype** – If *True*, the event's data can be a subtype of the type specified in *dtype*. If *False*, it must be exactly the type in *dtype*.

- **cast** – If *True*, and if the event's data is not the expected type, then an attempt is made to cast it to *dtype*. If *False*, no attempt is made to cast the event's data.

evaluate(*event: BoboEvent*, *history: BoboHistory*) → bool

Parameters

- **event** – The event used for evaluation.
- **history** – The history of currently accepted events.

Returns

True if predicate is satisfied; *False* otherwise.

Exceptions

BoboPredicateError	A predicate error.

bobocep.cep.phenom.pattern.predicate.BoboPredicateError

exception bobocep.cep.phenom.pattern.predicate.**BoboPredicateError**

A predicate error.

bobocep.cep.phenom.phenom

A phenomenon.

Classes

BoboPhenomenon(name, patterns[, action, ...])	A phenomenon, satisfied by patterns of events, which facilitates the generating of complex events

bobocep.cep.phenom.phenom.BoboPhenomenon

class bobocep.cep.phenom.phenom.**BoboPhenomenon**(*name: str*, *patterns: List[BoboPattern]*, *action: BoboAction | None = None*, *datagen: Callable | None = None*, *retain: bool = True*)

Bases: `object`

A phenomenon, satisfied by patterns of events, which facilitates the generating of complex events

__init__(*name: str*, *patterns: List[BoboPattern]*, *action: BoboAction | None = None*, *datagen: Callable | None = None*, *retain: bool = True*)

Parameters

- **name** – Phenomenon name.
- **patterns** – Phenomenon patterns.
- **action** – Phenomenon action.

- **datagen** – Phenomenon datagen.
- **retain** – If *True*, retains datagen callable as an object variable to prevent garbage collection of it.

property action: *BoboAction* **| None**

Returns
Phenomenon action, or *None*.

property datagen: Callable | None

Returns
Phenomenon datagen, or *None*.

property name: str

Returns
Phenomenon name.

property patterns: Tuple[*BoboPattern***, ...]**

Returns
Phenomenon patterns.

property retain: bool

Returns
True if retains datagen callable; False otherwise.

Exceptions

BoboPhenomenonError	A phenomenon error.

bobocep.cep.phenom.phenom.BoboPhenomenonError

exception bobocep.cep.phenom.phenom.**BoboPhenomenonError**

A phenomenon error.

10.1.3 bobocep.dist

Distributed imports.

Modules

bobocep.dist.crypto	Crypto imports.
bobocep.dist.device	Devices on the network.
bobocep.dist.devman	Device manager.
bobocep.dist.dist	Distributed complex event processing.
bobocep.dist.pubsub	Distributed publish-subscribe classes.
bobocep.dist.tcp	Distributed *BoboCEP* via TCP.

bobocep.dist.crypto

Crypto imports.

Modules

bobocep.dist.crypto.aes	AES encryption for distributed traffic.
bobocep.dist.crypto.crypto	Encryption for distributed *BoboCEP*.

bobocep.dist.crypto.aes

AES encryption for distributed traffic.

Classes

BoboDistributedCryptoAES(aes_key[, ...])	AES encryption in GCM mode.

bobocep.dist.crypto.aes.BoboDistributedCryptoAES

class bobocep.dist.crypto.aes.**BoboDistributedCryptoAES**(*aes_key: str*, *nonce_length: int = 16*, *mac_length: int = 16*)

Bases: *BoboDistributedCrypto*

AES encryption in GCM mode. Data are encrypted using either AES-128, AES-192, or AES-256 encryption.

__init__(*aes_key: str*, *nonce_length: int = 16*, *mac_length: int = 16*)

Parameters

- **aes_key** – The AES key to use for encryption.
- **nonce_length** – The nonce length.
- **mac_length** – The MAC length.

decrypt(*msg_bytes: bytes*) → str

Parameters
msg_bytes – Incoming bytes.

Returns
Incoming JSON string with other data from transit.

encrypt(*msg_str: str*) → bytes

Parameters
msg_str – Wraps JSON string in other data for transit.

Returns
The bytes to transmit.

end_bytes() → bytes

Returns
Bytes used to signify the end of every encrypted payload.

min_length() → int

Returns
The minimum possible length of an encrypted payload, in number of bytes.

bobocep.dist.crypto.crypto

Encryption for distributed *BoboCEP*.

Classes

BoboDistributedCrypto()	Abstract class for distributed crypto.

bobocep.dist.crypto.crypto.BoboDistributedCrypto

class bobocep.dist.crypto.crypto.**BoboDistributedCrypto**

Bases: ABC

Abstract class for distributed crypto.

__init__()

abstract **decrypt**(*msg_bytes: bytes*) → str

Parameters
msg_bytes – Message to decrypt.

Returns
Decrypted message.

abstract **encrypt**(*msg_str: str*) → bytes

Parameters
msg_str – Message to encrypt.

Returns
Encrypted message.

abstract **end_bytes**() → bytes

Returns
Bytes used to signify the end of every encrypted payload.

abstract **min_length**() → int

Returns
The minimum possible length of an encrypted payload, in number of bytes.

Exceptions

BoboDistributedCryptoError	A distributed crypto error.

bobocep.dist.crypto.crypto.BoboDistributedCryptoError

exception bobocep.dist.crypto.crypto.**BoboDistributedCryptoError**

A distributed crypto error.

bobocep.dist.device

Devices on the network.

Classes

BoboDevice(addr, port, urn, id_key)	Contains information about a BoboCEP instance on the network.

bobocep.dist.device.BoboDevice

class bobocep.dist.device.**BoboDevice**(*addr: str*, *port: int*, *urn: str*, *id_key: str*)

Bases: object

Contains information about a BoboCEP instance on the network.

__init__(*addr: str*, *port: int*, *urn: str*, *id_key: str*)

Parameters

- **addr** – Device address.
- **port** – Device port.
- **urn** – Device URN.
- **id_key** – Device ID key.

property addr: str

Returns
Device address.

property id_key: str

Returns
Device ID key.

property port: int

Returns
Device port.

property urn: str

Returns
Device URN.

Exceptions

BoboDeviceError	A device error.

bobocep.dist.device.BoboDeviceError

exception bobocep.dist.device.**BoboDeviceError**

A device error.

bobocep.dist.devman

Device manager.

Classes

BoboDeviceManager(device, flag_reset)	Manages information about a BoboCEP device on the network.

bobocep.dist.devman.BoboDeviceManager

class bobocep.dist.devman.**BoboDeviceManager**(*device: BoboDevice*, *flag_reset: bool*)

Bases: object

Manages information about a BoboCEP device on the network.

__init__(*device: BoboDevice*, *flag_reset: bool*)

Parameters

- **device** – The device to manage.
- **flag_reset** – If *True*, it indicates to the BoboDistributed instance that the next message to the device should set a flag to reset its data on the sending device; 'False' indicates that no flag should be set.

property addr: str

Returns
Device address.

append_stash(*completed: List[BoboRunSerial]*, *halted: List[BoboRunSerial]*, *updated: List[BoboRunSerial]*) → None

Append runs to stash.

Parameters

- **completed** – Completed runs to append.
- **halted** – Halted runs to append.
- **updated** – Updated runs to append.

`clear_last()` → None

Clears both the last communication and last attempted communication time by setting them both to 0.

`clear_stash()` → None

Removes all items in the stash.

`property flag_reset: bool`

Returns
True if reset flag should be set; *False* otherwise.

`property id_key: str`

Returns
Device ID key.

`property last_attempt: int`

Returns
Last attempted communication with the device.

`property last_comms: int`

Returns
Time of last communication with device.

`property port: int`

Returns
Device port.

`size_stash()` → int

Returns
The stash size, equal to all completed, halted, and updated runs in the stash.

`stash()` → Tuple[List[*BoboRunSerial*], List[*BoboRunSerial*], List[*BoboRunSerial*]]

Returns
The device's stash.

`property urn: str`

Returns
Device URN.

bobocep.dist.dist

Distributed complex event processing.

Classes

BoboDistributed()	Distributed *BoboCEP*.

bobocep.dist.dist.BoboDistributed

class bobocep.dist.dist.**BoboDistributed**

Bases: ABC

Distributed *BoboCEP*.

__init__()

abstract run() → None

Runs distributed.

Exceptions

BoboDistributedError	A distributed error.
BoboDistributedJSONDecodeError	A distributed JSON decode error.
BoboDistributedJSONEncodeError	A distributed JSON encode error.
BoboDistributedJSONError	A distributed JSON error.
BoboDistributedSystemError	A distributed system error.
BoboDistributedTimeoutError	A distributed timeout error.

bobocep.dist.dist.BoboDistributedError

exception bobocep.dist.dist.**BoboDistributedError**

A distributed error.

bobocep.dist.dist.BoboDistributedJSONDecodeError

exception bobocep.dist.dist.**BoboDistributedJSONDecodeError**

A distributed JSON decode error.

bobocep.dist.dist.BoboDistributedJSONEncodeError

exception bobocep.dist.dist.**BoboDistributedJSONEncodeError**

A distributed JSON encode error.

bobocep.dist.dist.BoboDistributedJSONError

exception bobocep.dist.dist.**BoboDistributedJSONError**

A distributed JSON error.

bobocep.dist.dist.BoboDistributedSystemError

exception bobocep.dist.dist.**BoboDistributedSystemError**

A distributed system error.

bobocep.dist.dist.BoboDistributedTimeoutError

exception bobocep.dist.dist.**BoboDistributedTimeoutError**

A distributed timeout error.

bobocep.dist.pubsub

Distributed publish-subscribe classes.

Classes

BoboDistributedPublisher()	A distributed publisher interface.
BoboDistributedSubscriber()	A distributed subscriber interface.

bobocep.dist.pubsub.BoboDistributedPublisher

class bobocep.dist.pubsub.**BoboDistributedPublisher**

Bases: ABC

A distributed publisher interface.

__init__()

abstract subscribe(*subscriber:* BoboDistributedSubscriber) → None

Parameters

subscriber – Subscriber to distributed.

bobocep.dist.pubsub.BoboDistributedSubscriber

class bobocep.dist.pubsub.**BoboDistributedSubscriber**

Bases: ABC

A distributed subscriber interface.

__init__()

abstract on_distributed_update(*completed: List[BoboRunSerial]*, *halted: List[BoboRunSerial]*, *updated: List[BoboRunSerial]*) → None

Parameters

- **completed** – Completed runs.
- **halted** – Halted runs.
- **updated** – Updated runs.

bobocep.dist.tcp

Distributed *BoboCEP* via TCP.

Classes

BoboDistributedTCP(urn, decider, devices, crypto)	An implementation of distributed BoboCEP that uses TCP for data transmission across the network.

bobocep.dist.tcp.BoboDistributedTCP

class bobocep.dist.tcp.**BoboDistributedTCP**(*urn: str*, *decider: BoboDeciderPublisher*, *devices: List[BoboDevice]*, *crypto: BoboDistributedCrypto*, *max_size_incoming: int = 0*, *max_size_outgoing: int = 0*, *period_ping: int = 30*, *period_resync: int = 60*, *attempt_stash: int = 5*, *attempt_ping: int = 5*, *attempt_resync: int = 10*, *max_listen: int = 3*, *timeout_accept: int = 3*, *timeout_connect: int = 3*, *timeout_send: int = 3*, *timeout_receive: int = 3*, *recv_bytes: int = 2048*, *flag_reset: bool = True*)

Bases: *BoboDistributed*, *BoboDistributedPublisher*, *BoboDeciderSubscriber*

An implementation of distributed BoboCEP that uses TCP for data transmission across the network.

__init__(*urn: str*, *decider: BoboDeciderPublisher*, *devices: List[BoboDevice]*, *crypto: BoboDistributedCrypto*, *max_size_incoming: int = 0*, *max_size_outgoing: int = 0*, *period_ping: int = 30*, *period_resync: int = 60*, *attempt_stash: int = 5*, *attempt_ping: int = 5*, *attempt_resync: int = 10*, *max_listen: int = 3*, *timeout_accept: int = 3*, *timeout_connect: int = 3*, *timeout_send: int = 3*, *timeout_receive: int = 3*, *recv_bytes: int = 2048*, *flag_reset: bool = True*)

Parameters

- **urn** – A URN that is unique across devices in the network.

- **decider** – The Decider used in the local engine.
- **devices** – Devices in the network (including this device).
- **crypto** – Encryption to use for message exchange.
- **max_size_incoming** – Max queue size for incoming data. Default: 0 (unbounded).
- **max_size_outgoing** – Max queue size for outgoing data. Default: 0 (unbounded).
- **period_ping** – Period of inactivity from another device to warrant pinging the device, in seconds. Default: 30.
- **period_resync** – Period of inactivity from another device to warrant resyncing with the device, in seconds. Default: 60.
- **attempt_stash** – How frequently to attempt to send the sync stash if the stash is not empty.
- **attempt_ping** – How frequently to ping another device if it is within the ping period, in seconds. Default: 10.
- **attempt_resync** – How frequently to resync with another device if it is within the resync period, in seconds. Default: 10.
- **max_listen** – Max number of incoming connections to listen for at a given time, in seconds. Default: 3.
- **timeout_accept** – Timeout for accepting a new incoming connection, in seconds. Default: 3.
- **timeout_connect** – Timeout for connecting to a client when sending data, in seconds. Default: 3.
- **timeout_send** – Timeout for sending data, in seconds. Default: 3.
- **timeout_receive** – Timeout for receiving data, in seconds. Default: 3.
- **recv_bytes** – Number of bytes to receive at a time when receiving data. Default: 2048.
- **flag_reset** – If *True*, the RESET flag is set to indicate to external devices that it should reset its data on this device, which will trigger a resync.

close() → None

Closes the distributed instance.

is_closed() → bool

Returns

True if distributed is closed; *False* otherwise.

join() → None

Joins with the incoming and outgoing threads.

on_decider_update(*completed: List[*BoboRunSerial*]*, *halted: List[*BoboRunSerial*]*, *updated: List[*BoboRunSerial*]*, *local: bool*) → None

Parameters

- **completed** – Locally completed runs.
- **halted** – Locally halted runs.
- **updated** – Locally updated runs.

- **local** – *True* if the Decider update occurred locally; *False* if the update occurred on a remote (distributed) instance.

run() → None

Runs the distributed instance.

size_incoming() → int

Returns
Size of incoming queue.

size_outgoing() → int

Returns
Size of outgoing queue

subscribe(*subscriber:* BoboDistributedSubscriber) → None

Parameters
subscriber – Subscriber to the distributed instance.

10.1.4 bobocep.setup

Setup imports.

Modules

bobocep.setup.setup	Tools to help with *BoboCEP* setup.
bobocep.setup.simple	Simple setup.

bobocep.setup.setup

Tools to help with *BoboCEP* setup.

Classes

BoboSetup()	A setup.

bobocep.setup.setup.BoboSetup

class bobocep.setup.setup.**BoboSetup**

Bases: ABC

A setup.

__init__()

abstract generate() → Any

Returns
Relevant setup data.

Exceptions

BoboSetupError	A setup error.

bobocep.setup.setup.BoboSetupError

exception bobocep.setup.setup.**BoboSetupError**

A setup error.

bobocep.setup.simple

Simple setup.

Classes

BoboSetupSimple(phenomena, handler[, ...])	A simple setup to make configuration easier.
BoboSetupSimpleDistributed(phenomena, ...[, ...])	A simple setup to make distributed configuration easier.

bobocep.setup.simple.BoboSetupSimple

class bobocep.setup.simple.**BoboSetupSimple**(*phenomena: List[*BoboPhenomenon*]*, *handler:* BoboActionHandler, *validator:* BoboValidator | *None = None*, *gen_event:* BoboGenEvent | *None = None*, *urn: str* | *None = None*)

Bases: *BoboSetup*

A simple setup to make configuration easier.

__init__(*phenomena: List[*BoboPhenomenon*]*, *handler:* BoboActionHandler, *validator:* BoboValidator | *None = None*, *gen_event:* BoboGenEvent | *None = None*, *urn: str* | *None = None*)

Parameters

- **phenomena** – A list of phenomena.
- **handler** – An action handler.
- **validator** – A data validator for the engine's Receiver task. Default: BoboValidatorAll.
- **gen_event** – An event generator. Default: None.
- **urn** – A URN for ID generation.

generate() → *BoboEngine*

Returns

The CEP engine.

bobocep.setup.simple.BoboSetupSimpleDistributed

class bobocep.setup.simple.**BoboSetupSimpleDistributed**(*phenomena: List[*BoboPhenomenon*]*, *handler:* BoboActionHandler, *urn: str*, *devices: List[*BoboDevice*]*, *aes_key: str*, *validator:* BoboValidatorJSONable | *None* = *None*, *gen_event:* BoboGenEvent | *None* = *None*)

Bases: *BoboSetup*

A simple setup to make distributed configuration easier.

__init__(*phenomena: List[*BoboPhenomenon*]*, *handler:* BoboActionHandler, *urn: str*, *devices: List[*BoboDevice*]*, *aes_key: str*, *validator:* BoboValidatorJSONable | *None* = *None*, *gen_event:* BoboGenEvent | *None* = *None*)

Parameters

- **phenomena** – A list of phenomena.
- **handler** – An action handler.
- **urn** – A URN that is unique across devices in the network.
- **devices** – Devices in the network (including this device).
- **aes_key** – The AES key to use for encryption.
- **validator** – A data validator for the engine's Receiver task. Default: BoboValidatorAll.
- **gen_event** – An event generator. Default: None.

generate() → Tuple[*BoboEngine*, *BoboDistributedTCP*]

Returns

The CEP engine and distributed TCP instance with AES encryption.

CHAPTER

ELEVEN

DEVELOPER GUIDE

11.1 Dependencies

You will need to install the core BoboCEP requirements from both `requirements.txt` and its additional development requirements from `requirements-dev.txt`. For example:

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

11.2 Development Tools

BoboCEP uses GitHub Actions for *Continuous Integration* (CI) and *Continuous Deployment* (CD). It uses two YAML scripts to trigger the respective action workflows, namely:

1. `.github/workflows/cicd.yml` for CI/CD tasks, including: linting, type checking, code coverage, documentation coverage, and deployment to PyPI.
2. `.github/workflows/security.yml` for security checks.

These scripts are triggered on a push to the `main` branch. The security script also runs periodically.

It is recommended that you run the individual CI/CD tasks manually before committing. These are discussed next.

11.2.1 Code Linting

`flake8` is used for code linting. Run the following two commands to lint BoboCEP and its test suite, respectively.

```
flake8 ./bobocep --count --select=E9,F63,F7,F82 --show-source --statistics
flake8 ./tests --count --select=E9,F63,F7,F82 --show-source --statistics
```

11.2.2 Code Testing and Coverage

coverage is used for code testing and coverage. Results are uploaded to Code Climate. Run the following command to test BoboCEP.

```
coverage run -m pytest tests
```

The coverage configuration can be found in .coveragerc. GitHub Actions additionally enforces a minimum coverage of 100%. You can check that this requirement has been satisfied using the following.

```
coverage report --fail-under=100
```

Note: If you are unable to achieve 100% coverage with your code contribution, you can omit code from testing in .coveragerc.

You can locally inspect the code coverage with an HTML output by running the following.

```
coverage html
```

11.2.3 Documentation

Documentation is built using sphinx and is deployed via Read the Docs. You can compile documentation locally via the following.

```
cd docs
make html
```

Or, for Windows (PowerShell).

```
cd docs
.\make.bat html
```

11.2.4 Documentation Coverage

interrogate is used for testing and code coverage. Run the following command to check BoboCEP documentation coverage. It requires a minimum documentation coverage of 100%.

```
interrogate -vv bobocep --fail-under 100
```

11.2.5 Type Checking

mypy is used for type checking. Run the following two commands to check BoboCEP and its test suite, respectively.

```
mypy ./bobocep
mypy ./tests
```

11.2.6 Versioning

BoboCEP uses Semantic Versioning and the bump2version tool for editing the software version. See here for more information. The major, minor, or patch components of the version number can be changed with the following:

```
bump2version patch
```

CHAPTER

TWELVE

CONTRIBUTING

If you would like to contribute to the BoboCEP project, please consult the CONTRIBUTING.md document for more information.

Here are other ways through which you can contribute.

- Submit an issue for bug reports and feature requests.
- Report any security vulnerabilities that you become aware of.
- Donate to the project via Ko-fi.

BIBLIOGRAPHY

[CM2012] Cugola, G., & Margara, A. (2012). Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, *44*(3), 15.

[P2023] Power, A. (2023). BoboCEP: a Fault-Tolerant Complex Event Processing Engine for Edge Computing in Internet of Things. *Journal of Open Source Software*.

[PK2020] Power, A., & Kotonya, G. (2020). BoboCEP: Distributed Complex Event Processing for Resilient Fault-Tolerance Support in IoT. *IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService)*.

PYTHON MODULE INDEX

b

bobocep, 43
bobocep.bobocep, 43
bobocep.cep, 45
bobocep.cep.action, 45
bobocep.cep.action.action, 45
bobocep.cep.action.common, 46
bobocep.cep.action.common.multi, 46
bobocep.cep.action.handler, 48
bobocep.cep.engine, 52
bobocep.cep.engine.decider, 52
bobocep.cep.engine.decider.decider, 53
bobocep.cep.engine.decider.pubsub, 55
bobocep.cep.engine.decider.run, 56
bobocep.cep.engine.decider.runserial, 58
bobocep.cep.engine.engine, 60
bobocep.cep.engine.forwarder, 62
bobocep.cep.engine.forwarder.forwarder, 62
bobocep.cep.engine.forwarder.pubsub, 63
bobocep.cep.engine.producer, 64
bobocep.cep.engine.producer.producer, 65
bobocep.cep.engine.producer.pubsub, 66
bobocep.cep.engine.receiver, 67
bobocep.cep.engine.receiver.pubsub, 67
bobocep.cep.engine.receiver.receiver, 68
bobocep.cep.engine.receiver.validator, 70
bobocep.cep.engine.task, 72
bobocep.cep.event, 73
bobocep.cep.event.action, 73
bobocep.cep.event.complex, 75
bobocep.cep.event.event, 77
bobocep.cep.event.factory, 78
bobocep.cep.event.history, 79
bobocep.cep.event.simple, 80
bobocep.cep.gen, 82
bobocep.cep.gen.event, 82
bobocep.cep.gen.event_id, 84
bobocep.cep.gen.timestamp, 84
bobocep.cep.phenom, 85
bobocep.cep.phenom.pattern, 86
bobocep.cep.phenom.pattern.builder, 86
bobocep.cep.phenom.pattern.pattern, 89
bobocep.cep.phenom.pattern.predicate, 91
bobocep.cep.phenom.phenom, 93
bobocep.dist, 94
bobocep.dist.crypto, 95
bobocep.dist.crypto.aes, 95
bobocep.dist.crypto.crypto, 96
bobocep.dist.device, 97
bobocep.dist.devman, 98
bobocep.dist.dist, 100
bobocep.dist.pubsub, 101
bobocep.dist.tcp, 102
bobocep.setup, 104
bobocep.setup.setup, 104
bobocep.setup.simple, 105

INDEX

Symbols

__init__() (*bobocep.bobocep.BoboJSONable method*), 44
__init__() (*bobocep.cep.action.action.BoboAction method*), 45
__init__() (*bobocep.cep.action.common.multi.BoboActionMulti method*), 47
__init__() (*bobocep.cep.action.common.multi.BoboActionMultiSequential method*), 47
__init__() (*bobocep.cep.action.handler.BoboActionHandler method*), 48
__init__() (*bobocep.cep.action.handler.BoboActionHandlerBlocking method*), 49
__init__() (*bobocep.cep.action.handler.BoboActionHandlerMultiprocessing method*), 50
__init__() (*bobocep.cep.action.handler.BoboActionHandlerMultithreading method*), 50
__init__() (*bobocep.cep.action.handler.BoboHandlerResponse method*), 51
__init__() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 53
__init__() (*bobocep.cep.engine.decider.pubsub.BoboDeciderPublisher method*), 55
__init__() (*bobocep.cep.engine.decider.pubsub.BoboDeciderSubscriber method*), 55
__init__() (*bobocep.cep.engine.decider.run.BoboRun method*), 56
__init__() (*bobocep.cep.engine.decider.runserial.BoboRunSerial method*), 58
__init__() (*bobocep.cep.engine.engine.BoboEngine method*), 60
__init__() (*bobocep.cep.engine.forwarder.forwarder.BoboForwarder method*), 62
__init__() (*bobocep.cep.engine.forwarder.pubsub.BoboForwarderPublisher method*), 64
__init__() (*bobocep.cep.engine.forwarder.pubsub.BoboForwarderSubscriber method*), 64
__init__() (*bobocep.cep.engine.producer.producer.BoboProducer method*), 65
__init__() (*bobocep.cep.engine.producer.pubsub.BoboProducerPublisher method*), 66
__init__() (*bobocep.cep.engine.producer.pubsub.BoboProducerSubscriber method*), 67
__init__() (*bobocep.cep.engine.receiver.pubsub.BoboReceiverPublisher method*), 67
__init__() (*bobocep.cep.engine.receiver.pubsub.BoboReceiverSubscriber method*), 68
__init__() (*bobocep.cep.engine.receiver.receiver.BoboReceiver method*), 68
__init__() (*bobocep.cep.engine.receiver.validator.BoboValidator method*), 70
__init__() (*bobocep.cep.engine.receiver.validator.BoboValidatorAll method*), 70
__init__() (*bobocep.cep.engine.receiver.validator.BoboValidatorJSONSc method*), 71
__init__() (*bobocep.cep.engine.receiver.validator.BoboValidatorJSONab method*), 71
__init__() (*bobocep.cep.engine.receiver.validator.BoboValidatorType method*), 71
__init__() (*bobocep.cep.engine.task.BoboEngineTask method*), 72
__init__() (*bobocep.cep.event.action.BoboEventAction method*), 73
__init__() (*bobocep.cep.event.complex.BoboEventComplex method*), 75
__init__() (*bobocep.cep.event.event.BoboEvent method*), 77
__init__() (*bobocep.cep.event.factory.BoboEventFactory method*), 78
__init__() (*bobocep.cep.event.history.BoboHistory method*), 79
__init__() (*bobocep.cep.event.simple.BoboEventSimple method*), 81
__init__() (*bobocep.cep.gen.event.BoboGenEvent method*), 82
__init__() (*bobocep.cep.gen.event.BoboGenEventNone method*), 83
__init__() (*bobocep.cep.gen.event.BoboGenEventTime method*), 83
__init__() (*bobocep.cep.gen.event_id.BoboGenEventID method*), 84
__init__() (*bobocep.cep.gen.event_id.BoboGenEventIDUnique method*), 84
__init__() (*bobocep.cep.gen.timestamp.BoboGenTimestamp*

method), 85
__init__() (bobocep.cep.gen.timestamp.BoboGenTimestampEpoch method), 85
__init__() (bobocep.cep.phenom.pattern.builder.BoboPatternBuilder method), 86
__init__() (bobocep.cep.phenom.pattern.pattern.BoboPattern method), 89
__init__() (bobocep.cep.phenom.pattern.pattern.BoboPatternBlock method), 90
__init__() (bobocep.cep.phenom.pattern.predicate.BoboPredicate method), 91
__init__() (bobocep.cep.phenom.pattern.predicate.BoboPredicateCall method), 92
__init__() (bobocep.cep.phenom.pattern.predicate.BoboPredicateCallType method), 92
__init__() (bobocep.cep.phenom.phenom.BoboPhenomenon method), 93
__init__() (bobocep.dist.crypto.aes.BoboDistributedCryptoAES method), 95
__init__() (bobocep.dist.crypto.crypto.BoboDistributedCrypto method), 96
__init__() (bobocep.dist.device.BoboDevice method), 97
__init__() (bobocep.dist.devman.BoboDeviceManager method), 98
__init__() (bobocep.dist.dist.BoboDistributed method), 100
__init__() (bobocep.dist.pubsub.BoboDistributedPublisher method), 101
__init__() (bobocep.dist.pubsub.BoboDistributedSubscriber method), 102
__init__() (bobocep.dist.tcp.BoboDistributedTCP method), 102
__init__() (bobocep.setup.setup.BoboSetup method), 104
__init__() (bobocep.setup.simple.BoboSetupSimple method), 105
__init__() (bobocep.setup.simple.BoboSetupSimpleDistributed method), 106

A

action (bobocep.cep.phenom.phenom.BoboPhenomenon property), 94
action_name (bobocep.cep.action.handler.BoboHandlerResponse attribute), 51
action_name (bobocep.cep.event.action.BoboEventAction property), 74
add_data() (bobocep.cep.engine.receiver.receiver.BoboReceiver method), 68
addr (bobocep.dist.device.BoboDevice property), 97
addr (bobocep.dist.devman.BoboDeviceManager property), 98
all_events() (bobocep.cep.event.history.BoboHistory method), 79
all_groups() (bobocep.cep.event.history.BoboHistory method), 79
all_runs() (bobocep.cep.engine.decider.decider.BoboDecider method), 53
append_stash() (bobocep.dist.devman.BoboDeviceManager method), 98

B

block_index (bobocep.cep.engine.decider.run.BoboRun property), 56
block_index (bobocep.cep.engine.decider.runserial.BoboRunSerial property), 59
blocks (bobocep.cep.phenom.pattern.pattern.BoboPattern property), 89
BoboAction (class in bobocep.cep.action.action), 45
BoboActionError, 46
BoboActionHandler (class in bobocep.cep.action.handler), 48
BoboActionHandlerBlocking (class in bobocep.cep.action.handler), 49
BoboActionHandlerError, 52
BoboActionHandlerMultiprocessing (class in bobocep.cep.action.handler), 50
BoboActionHandlerMultithreading (class in bobocep.cep.action.handler), 50
BoboActionMulti (class in bobocep.cep.action.common.multi), 47
BoboActionMultiSequential (class in bobocep.cep.action.common.multi), 47
bobocep
module, 43
bobocep.bobocep
module, 43
bobocep.cep
module, 45
bobocep.cep.action
module, 45
bobocep.cep.action.action
module, 45
bobocep.cep.action.common
module, 46
bobocep.cep.action.common.multi
module, 46
bobocep.cep.action.handler
module, 48
bobocep.cep.engine
module, 52
bobocep.cep.engine.decider
module, 52
bobocep.cep.engine.decider.decider
module, 53
bobocep.cep.engine.decider.pubsub
module, 55

bobocep.cep.engine.decider.run
module, 56
bobocep.cep.engine.decider.runserial
module, 58
bobocep.cep.engine.engine
module, 60
bobocep.cep.engine.forwarder
module, 62
bobocep.cep.engine.forwarder.forwarder
module, 62
bobocep.cep.engine.forwarder.pubsub
module, 63
bobocep.cep.engine.producer
module, 64
bobocep.cep.engine.producer.producer
module, 65
bobocep.cep.engine.producer.pubsub
module, 66
bobocep.cep.engine.receiver
module, 67
bobocep.cep.engine.receiver.pubsub
module, 67
bobocep.cep.engine.receiver.receiver
module, 68
bobocep.cep.engine.receiver.validator
module, 70
bobocep.cep.engine.task
module, 72
bobocep.cep.event
module, 73
bobocep.cep.event.action
module, 73
bobocep.cep.event.complex
module, 75
bobocep.cep.event.event
module, 77
bobocep.cep.event.factory
module, 78
bobocep.cep.event.history
module, 79
bobocep.cep.event.simple
module, 80
bobocep.cep.gen
module, 82
bobocep.cep.gen.event
module, 82
bobocep.cep.gen.event_id
module, 84
bobocep.cep.gen.timestamp
module, 84
bobocep.cep.phenom
module, 85
bobocep.cep.phenom.pattern
module, 86
bobocep.cep.phenom.pattern.builder
module, 86
bobocep.cep.phenom.pattern.pattern
module, 89
bobocep.cep.phenom.pattern.predicate
module, 91
bobocep.cep.phenom.phenom
module, 93
bobocep.dist
module, 94
bobocep.dist.crypto
module, 95
bobocep.dist.crypto.aes
module, 95
bobocep.dist.crypto.crypto
module, 96
bobocep.dist.device
module, 97
bobocep.dist.devman
module, 98
bobocep.dist.dist
module, 100
bobocep.dist.pubsub
module, 101
bobocep.dist.tcp
module, 102
bobocep.setup
module, 104
bobocep.setup.setup
module, 104
bobocep.setup.simple
module, 105
BoboDecider (*class in bobocep.cep.engine.decider.decider*), 53
BoboDeciderError, 55
BoboDeciderPublisher (*class in bobocep.cep.engine.decider.pubsub*), 55
BoboDeciderSubscriber (*class in bobocep.cep.engine.decider.pubsub*), 55
BoboDevice (*class in bobocep.dist.device*), 97
BoboDeviceError, 98
BoboDeviceManager (*class in bobocep.dist.devman*), 98
BoboDistributed (*class in bobocep.dist.dist*), 100
BoboDistributedCrypto (*class in bobocep.dist.crypto.crypto*), 96
BoboDistributedCryptoAES (*class in bobocep.dist.crypto.aes*), 95
BoboDistributedCryptoError, 97
BoboDistributedError, 100
BoboDistributedJSONDecodeError, 100
BoboDistributedJSONEncodeError, 101
BoboDistributedJSONError, 101
BoboDistributedPublisher (*class in bobocep.dist.pubsub*), 101

BoboDistributedSubscriber (*class in bobocep.dist.pubsub*), 102
BoboDistributedSystemError, 101
BoboDistributedTCP (*class in bobocep.dist.tcp*), 102
BoboDistributedTimeoutError, 101
BoboEngine (*class in bobocep.cep.engine.engine*), 60
BoboEngineError, 61
BoboEngineTask (*class in bobocep.cep.engine.task*), 72
BoboEngineTaskError, 73
BoboError, 44
BoboEvent (*class in bobocep.cep.event.event*), 77
BoboEventAction (*class in bobocep.cep.event.action*), 73
BoboEventComplex (*class in bobocep.cep.event.complex*), 75
BoboEventError, 78
BoboEventFactory (*class in bobocep.cep.event.factory*), 78
BoboEventFactoryError, 79
BoboEventSimple (*class in bobocep.cep.event.simple*), 81
BoboForwarder (*class in bobocep.cep.engine.forwarder.forwarder*), 62
BoboForwarderError, 63
BoboForwarderPublisher (*class in bobocep.cep.engine.forwarder.pubsub*), 64
BoboForwarderSubscriber (*class in bobocep.cep.engine.forwarder.pubsub*), 64
BoboGenEvent (*class in bobocep.cep.gen.event*), 82
BoboGenEventID (*class in bobocep.cep.gen.event_id*), 84
BoboGenEventIDUnique (*class in bobocep.cep.gen.event_id*), 84
BoboGenEventNone (*class in bobocep.cep.gen.event*), 83
BoboGenEventTime (*class in bobocep.cep.gen.event*), 83
BoboGenTimestamp (*class in bobocep.cep.gen.timestamp*), 85
BoboGenTimestampEpoch (*class in bobocep.cep.gen.timestamp*), 85
BoboHandlerResponse (*class in bobocep.cep.action.handler*), 51
BoboHistory (*class in bobocep.cep.event.history*), 79
BoboJSONable (*class in bobocep.bobocep*), 44
BoboJSONableError, 44
BoboPattern (*class in bobocep.cep.phenom.pattern.pattern*), 89
BoboPatternBlock (*class in bobocep.cep.phenom.pattern.pattern*), 90
BoboPatternBlockError, 91
BoboPatternBuilder (*class in bobocep.cep.phenom.pattern.builder*), 86
BoboPatternBuilderError, 89
BoboPatternError, 91
BoboPhenomenon (*class in bobocep.cep.phenom.phenom*), 93
BoboPhenomenonError, 94
BoboPredicate (*class in bobocep.cep.phenom.pattern.predicate*), 91
BoboPredicateCall (*class in bobocep.cep.phenom.pattern.predicate*), 92
BoboPredicateCallType (*class in bobocep.cep.phenom.pattern.predicate*), 92
BoboPredicateError, 93
BoboProducer (*class in bobocep.cep.engine.producer.producer*), 65
BoboProducerError, 66
BoboProducerPublisher (*class in bobocep.cep.engine.producer.pubsub*), 66
BoboProducerSubscriber (*class in bobocep.cep.engine.producer.pubsub*), 67
BoboReceiver (*class in bobocep.cep.engine.receiver.receiver*), 68
BoboReceiverError, 70
BoboReceiverPublisher (*class in bobocep.cep.engine.receiver.pubsub*), 67
BoboReceiverSubscriber (*class in bobocep.cep.engine.receiver.pubsub*), 68
BoboRun (*class in bobocep.cep.engine.decider.run*), 56
BoboRunError, 58
BoboRunSerial (*class in bobocep.cep.engine.decider.runserial*), 58
BoboRunSerialError, 60
BoboSetup (*class in bobocep.setup.setup*), 104
BoboSetupError, 105
BoboSetupSimple (*class in bobocep.setup.simple*), 105
BoboSetupSimpleDistributed (*class in bobocep.setup.simple*), 106
BoboValidator (*class in bobocep.cep.engine.receiver.validator*), 70
BoboValidatorAll (*class in bobocep.cep.engine.receiver.validator*), 70
BoboValidatorError, 72
BoboValidatorJSONable (*class in bobocep.cep.engine.receiver.validator*), 71
BoboValidatorJSONSchema (*class in bobocep.cep.engine.receiver.validator*), 71
BoboValidatorType (*class in bobocep.cep.engine.receiver.validator*), 71

C

cast() (*bobocep.cep.event.action.BoboEventAction method*), 74
cast() (*bobocep.cep.event.complex.BoboEventComplex method*), 75
cast() (*bobocep.cep.event.event.BoboEvent method*), 77
cast() (*bobocep.cep.event.simple.BoboEventSimple method*), 81

clear_last() (*bobocep.dist.devman.BoboDeviceManager method*), 99
clear_stash() (*bobocep.dist.devman.BoboDeviceManager method*), 99
close() (*bobocep.cep.action.handler.BoboActionHandler method*), 48
close() (*bobocep.cep.action.handler.BoboActionHandlerBlocking method*), 49
close() (*bobocep.cep.action.handler.BoboActionHandlerMultiprocessing method*), 50
close() (*bobocep.cep.action.handler.BoboActionHandlerMultithreading method*), 51
close() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 53
close() (*bobocep.cep.engine.engine.BoboEngine method*), 61
close() (*bobocep.cep.engine.forwarder.forwarder.BoboForwarder method*), 62
close() (*bobocep.cep.engine.producer.producer.BoboProducer method*), 65
close() (*bobocep.cep.engine.receiver.receiver.BoboReceiver method*), 69
close() (*bobocep.cep.engine.task.BoboEngineTask method*), 72
close() (*bobocep.dist.tcp.BoboDistributedTCP method*), 103
complex_event (*bobocep.cep.action.handler.BoboHandlerResponse attribute*), 51
count() (*bobocep.cep.action.handler.BoboHandlerResponse method*), 51

D

data (*bobocep.cep.action.handler.BoboHandlerResponse attribute*), 51
data (*bobocep.cep.event.action.BoboEventAction property*), 74
data (*bobocep.cep.event.complex.BoboEventComplex property*), 76
data (*bobocep.cep.event.event.BoboEvent property*), 77
data (*bobocep.cep.event.simple.BoboEventSimple property*), 81
datagen (*bobocep.cep.phenom.phenom.BoboPhenomenon property*), 94
decider (*bobocep.cep.engine.engine.BoboEngine property*), 61
decrypt() (*bobocep.dist.crypto.aes.BoboDistributedCryptoAES method*), 95
decrypt() (*bobocep.dist.crypto.crypto.BoboDistributedCrypto method*), 96

E

encrypt() (*bobocep.dist.crypto.aes.BoboDistributedCryptoAES method*), 95
encrypt() (*bobocep.dist.crypto.crypto.BoboDistributedCrypto method*), 96
end_bytes() (*bobocep.dist.crypto.aes.BoboDistributedCryptoAES method*), 95
end_bytes() (*bobocep.dist.crypto.crypto.BoboDistributedCrypto method*), 96
evaluate() (*bobocep.cep.phenom.pattern.predicate.BoboPredicate method*), 91
evaluate() (*bobocep.cep.phenom.pattern.predicate.BoboPredicateCall method*), 92
evaluate() (*bobocep.cep.phenom.pattern.predicate.BoboPredicateCallTy method*), 93
event_id (*bobocep.cep.event.action.BoboEventAction property*), 74
event_id (*bobocep.cep.event.complex.BoboEventComplex property*), 76
event_id (*bobocep.cep.event.event.BoboEvent property*), 77
event_id (*bobocep.cep.event.simple.BoboEventSimple property*), 81
events (*bobocep.cep.event.history.BoboHistory property*), 79
execute() (*bobocep.cep.action.action.BoboAction method*), 45
execute() (*bobocep.cep.action.common.multi.BoboActionMulti method*), 47
execute() (*bobocep.cep.action.common.multi.BoboActionMultiSequential method*), 47

F

first() (*bobocep.cep.event.history.BoboHistory method*), 79
flag_reset (*bobocep.dist.devman.BoboDeviceManager property*), 99
followed_by() (*bobocep.cep.phenom.pattern.builder.BoboPatternBuilder method*), 86
followed_by_any() (*bobocep.cep.phenom.pattern.builder.BoboPatternBuilder method*), 86
forwarder (*bobocep.cep.engine.engine.BoboEngine property*), 61
from_json_dict() (*bobocep.bobocep.BoboJSONable static method*), 44
from_json_dict() (*bobocep.cep.engine.decider.runserial.BoboRunSerial static method*), 59
from_json_dict() (*bobocep.cep.event.action.BoboEventAction static method*), 74
from_json_dict() (*bobocep.cep.event.complex.BoboEventComplex static method*), 76

from_json_dict() (*bobocep.cep.event.event.BoboEvent static method*), 77
from_json_dict() (*bobocep.cep.event.history.BoboHistory static method*), 80
from_json_dict() (*bobocep.cep.event.simple.BoboEventSimple static method*), 81
from_json_str() (*bobocep.bobocep.BoboJSONable static method*), 44
from_json_str() (*bobocep.cep.engine.decider.runserial.BoboRunSerial static method*), 59
from_json_str() (*bobocep.cep.event.action.BoboEventAction static method*), 74
from_json_str() (*bobocep.cep.event.complex.BoboEventComplex static method*), 76
from_json_str() (*bobocep.cep.event.event.BoboEvent static method*), 77
from_json_str() (*bobocep.cep.event.factory.BoboEventFactory static method*), 78
from_json_str() (*bobocep.cep.event.history.BoboHistory static method*), 80
from_json_str() (*bobocep.cep.event.simple.BoboEventSimple static method*), 81

G

generate() (*bobocep.cep.gen.event_id.BoboGenEventID method*), 84
generate() (*bobocep.cep.gen.event_id.BoboGenEventIDUnique method*), 84
generate() (*bobocep.cep.gen.timestamp.BoboGenTimestamp method*), 85
generate() (*bobocep.cep.gen.timestamp.BoboGenTimestampEpoch method*), 85
generate() (*bobocep.cep.phenom.pattern.builder.BoboPatternBuilder method*), 87
generate() (*bobocep.setup.setup.BoboSetup method*), 104
generate() (*bobocep.setup.simple.BoboSetupSimple method*), 105
generate() (*bobocep.setup.simple.BoboSetupSimpleDistributed method*), 106
get_handler_response() (*bobocep.cep.action.handler.BoboActionHandler method*), 48
get_handler_response() (*bobocep.cep.action.handler.BoboActionHandlerBlocking method*), 49
get_handler_response() (*bobocep.cep.action.handler.BoboActionHandlerMultiprocessing method*), 50
get_handler_response() (*bobocep.cep.action.handler.BoboActionHandlerMultithreading method*), 51
group (*bobocep.cep.phenom.pattern.pattern.BoboPatternBlock property*), 90
group() (*bobocep.cep.event.history.BoboHistory method*), 80

H

halt() (*bobocep.cep.engine.decider.run.BoboRun method*), 57
haltcondition() (*bobocep.cep.phenom.pattern.builder.BoboPatternBuilder method*), 87
haltconditions (*bobocep.cep.phenom.pattern.pattern.BoboPattern property*), 89
handle() (*bobocep.cep.action.handler.BoboActionHandler method*), 48
handle() (*bobocep.cep.action.handler.BoboActionHandlerBlocking method*), 49
handle() (*bobocep.cep.action.handler.BoboActionHandlerMultiprocessing method*), 50
handle() (*bobocep.cep.action.handler.BoboActionHandlerMultithreading method*), 51
history (*bobocep.cep.engine.decider.runserial.BoboRunSerial property*), 59
history (*bobocep.cep.event.complex.BoboEventComplex property*), 76
history() (*bobocep.cep.engine.decider.run.BoboRun method*), 57

I

id_key (*bobocep.dist.device.BoboDevice property*), 97
id_key (*bobocep.dist.devman.BoboDeviceManager property*), 99
index() (*bobocep.cep.action.handler.BoboHandlerResponse method*), 51
is_closed() (*bobocep.cep.action.handler.BoboActionHandler method*), 48
is_closed() (*bobocep.cep.action.handler.BoboActionHandlerBlocking method*), 49
is_closed() (*bobocep.cep.action.handler.BoboActionHandlerMultiprocessing method*), 50
is_closed() (*bobocep.cep.action.handler.BoboActionHandlerMultithreading method*), 51
is_closed() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 53
is_closed() (*bobocep.cep.engine.engine.BoboEngine method*), 61

is_closed() (*bobocep.cep.engine.forwarder.forwarder.BoboForwarder method*), 63
is_closed() (*bobocep.cep.engine.producer.producer.BoboProducer method*), 65
is_closed() (*bobocep.cep.engine.receiver.receiver.BoboReceiver method*), 69
is_closed() (*bobocep.cep.engine.task.BoboEngineTask method*), 72
is_closed() (*bobocep.dist.tcp.BoboDistributedTCP method*), 103
is_complete() (*bobocep.cep.engine.decider.run.BoboRun method*), 57
is_halted() (*bobocep.cep.engine.decider.run.BoboRun method*), 57
is_valid() (*bobocep.cep.engine.receiver.validator.BoboValidator method*), 70
is_valid() (*bobocep.cep.engine.receiver.validator.BoboValidatorAll method*), 70
is_valid() (*bobocep.cep.engine.receiver.validator.BoboValidatorJSONable method*), 71
is_valid() (*bobocep.cep.engine.receiver.validator.BoboValidatorJSONSchema method*), 71
is_valid() (*bobocep.cep.engine.receiver.validator.BoboValidatorType method*), 71

J

join() (*bobocep.cep.action.handler.BoboActionHandlerMultiprocessing method*), 50
join() (*bobocep.cep.action.handler.BoboActionHandlerMultithreading method*), 51
join() (*bobocep.dist.tcp.BoboDistributedTCP method*), 103

L

last() (*bobocep.cep.event.history.BoboHistory method*), 80
last_attempt (*bobocep.dist.devman.BoboDeviceManager property*), 99
last_comms (*bobocep.dist.devman.BoboDeviceManager property*), 99
loop (*bobocep.cep.phenom.pattern.pattern.BoboPatternBlock property*), 90

M

maybe_generate() (*bobocep.cep.gen.event.BoboGenEvent method*), 82
maybe_generate() (*bobocep.cep.gen.event.BoboGenEventNone method*), 83
maybe_generate() (*bobocep.cep.gen.event.BoboGenEventTime method*), 83
min_length() (*bobocep.dist.crypto.aes.BoboDistributedCryptoAES method*), 96
min_length() (*bobocep.dist.crypto.crypto.BoboDistributedCrypto method*), 96
module
bobocep, 43
bobocep.bobocep, 43
bobocep.cep, 45
bobocep.cep.action, 45
bobocep.cep.action.action, 45
bobocep.cep.action.common, 46
bobocep.cep.action.common.multi, 46
bobocep.cep.action.handler, 48
bobocep.cep.engine, 52
bobocep.cep.engine.decider, 52
bobocep.cep.engine.decider.decider, 53
bobocep.cep.engine.decider.pubsub, 55
bobocep.cep.engine.decider.run, 56
bobocep.cep.engine.decider.runserial, 58
bobocep.cep.engine.engine, 60
bobocep.cep.engine.forwarder, 62
bobocep.cep.engine.forwarder.forwarder, 62
bobocep.cep.engine.forwarder.pubsub, 63
bobocep.cep.engine.producer, 64
bobocep.cep.engine.producer.producer, 65
bobocep.cep.engine.producer.pubsub, 66
bobocep.cep.engine.receiver, 67
bobocep.cep.engine.receiver.pubsub, 67
bobocep.cep.engine.receiver.receiver, 68
bobocep.cep.engine.receiver.validator, 70
bobocep.cep.engine.task, 72
bobocep.cep.event, 73
bobocep.cep.event.action, 73
bobocep.cep.event.complex, 75
bobocep.cep.event.event, 77
bobocep.cep.event.factory, 78
bobocep.cep.event.history, 79
bobocep.cep.event.simple, 80
bobocep.cep.gen, 82
bobocep.cep.gen.event, 82
bobocep.cep.gen.event_id, 84
bobocep.cep.gen.timestamp, 84
bobocep.cep.phenom, 85
bobocep.cep.phenom.pattern, 86
bobocep.cep.phenom.pattern.builder, 86
bobocep.cep.phenom.pattern.pattern, 89
bobocep.cep.phenom.pattern.predicate, 91
bobocep.cep.phenom.phenom, 93
bobocep.dist, 94
bobocep.dist.crypto, 95
bobocep.dist.crypto.aes, 95
bobocep.dist.crypto.crypto, 96
bobocep.dist.device, 97

bobocep.dist.devman, 98
bobocep.dist.dist, 100
bobocep.dist.pubsub, 101
bobocep.dist.tcp, 102
bobocep.setup, 104
bobocep.setup.setup, 104
bobocep.setup.simple, 105

N

name (*bobocep.cep.action.action.BoboAction property*), 46
name (*bobocep.cep.action.common.multi.BoboActionMulti property*), 47
name (*bobocep.cep.action.common.multi.BoboActionMultiSequential property*), 48
name (*bobocep.cep.phenom.pattern.pattern.BoboPattern property*), 89
name (*bobocep.cep.phenom.phenom.BoboPhenomenon property*), 94
negated (*bobocep.cep.phenom.pattern.pattern.BoboPatternBlock property*), 90
next() (*bobocep.cep.phenom.pattern.builder.BoboPatternBuilder method*), 87
not_followed_by() (*bobocep.cep.phenom.pattern.builder.BoboPatternBuilder method*), 87
not_followed_by_any() (*bobocep.cep.phenom.pattern.builder.BoboPatternBuilder method*), 88
not_next() (*bobocep.cep.phenom.pattern.builder.BoboPatternBuilder method*), 88

O

on_decider_update() (*bobocep.cep.engine.decider.pubsub.BoboDeciderSubscriber method*), 55
on_decider_update() (*bobocep.cep.engine.producer.producer.BoboProducer method*), 65
on_decider_update() (*bobocep.dist.tcp.BoboDistributedTCP method*), 103
on_distributed_update() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 53
on_distributed_update() (*bobocep.dist.pubsub.BoboDistributedSubscriber method*), 102
on_forwarder_update() (*bobocep.cep.engine.forwarder.pubsub.BoboForwarderSubscriber method*), 64
on_forwarder_update() (*bobocep.cep.engine.receiver.receiver.BoboReceiver method*), 69
on_producer_update() (*bobocep.cep.engine.forwarder.forwarder.BoboForwarder method*), 63
on_producer_update() (*bobocep.cep.engine.producer.pubsub.BoboProducerSubscriber method*), 67
on_producer_update() (*bobocep.cep.engine.receiver.receiver.BoboReceiver method*), 69
on_receiver_update() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 53
on_receiver_update() (*bobocep.cep.engine.receiver.pubsub.BoboReceiverSubscriber method*), 68
optional (*bobocep.cep.phenom.pattern.pattern.BoboPatternBlock property*), 90

P

pattern (*bobocep.cep.engine.decider.run.BoboRun property*), 57
pattern_name (*bobocep.cep.engine.decider.runserial.BoboRunSerial property*), 59
pattern_name (*bobocep.cep.event.action.BoboEventAction property*), 74
pattern_name (*bobocep.cep.event.complex.BoboEventComplex property*), 76
patterns (*bobocep.cep.phenom.phenom.BoboPhenomenon property*), 94
phenomena() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 54
phenomenon_name (*bobocep.cep.engine.decider.run.BoboRun property*), 57
phenomenon_name (*bobocep.cep.engine.decider.runserial.BoboRunSerial property*), 59
phenomenon_name (*bobocep.cep.event.action.BoboEventAction property*), 74
phenomenon_name (*bobocep.cep.event.complex.BoboEventComplex property*), 76
port (*bobocep.dist.device.BoboDevice property*), 97
port (*bobocep.dist.devman.BoboDeviceManager property*), 99
precondition() (*bobocep.cep.phenom.pattern.builder.BoboPatternBuilder method*), 88
preconditions (*bobocep.cep.phenom.pattern.pattern.BoboPattern property*), 89
predicates (*bobocep.cep.phenom.pattern.pattern.BoboPatternBlock property*), 90

process() (*bobocep.cep.engine.decider.run.BoboRun method*), 57
producer (*bobocep.cep.engine.engine.BoboEngine property*), 61

R

receiver (*bobocep.cep.engine.engine.BoboEngine property*), 61
retain (*bobocep.cep.phenom.phenom.BoboPhenomenon property*), 94
run() (*bobocep.cep.engine.engine.BoboEngine method*), 61
run() (*bobocep.dist.dist.BoboDistributed method*), 100
run() (*bobocep.dist.tcp.BoboDistributedTCP method*), 104
run_at() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 54
run_id (*bobocep.cep.engine.decider.run.BoboRun property*), 57
run_id (*bobocep.cep.engine.decider.runserial.BoboRunSerial property*), 59
runs_from() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 54

S

serialize() (*bobocep.cep.engine.decider.run.BoboRun method*), 57
set_block() (*bobocep.cep.engine.decider.run.BoboRun method*), 57
singleton (*bobocep.cep.phenom.pattern.pattern.BoboPattern property*), 90
size() (*bobocep.cep.action.handler.BoboActionHandler method*), 49
size() (*bobocep.cep.action.handler.BoboActionHandlerBlocking method*), 49
size() (*bobocep.cep.action.handler.BoboActionHandlerMultiprocessing method*), 50
size() (*bobocep.cep.action.handler.BoboActionHandlerMultithreading method*), 51
size() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 54
size() (*bobocep.cep.engine.forwarder.forwarder.BoboForwarder method*), 63
size() (*bobocep.cep.engine.producer.producer.BoboProducer method*), 65
size() (*bobocep.cep.engine.receiver.receiver.BoboReceiver method*), 69
size() (*bobocep.cep.engine.task.BoboEngineTask method*), 72
size() (*bobocep.cep.event.history.BoboHistory method*), 80
size_incoming() (*bobocep.dist.tcp.BoboDistributedTCP method*), 104
size_outgoing() (*bobocep.dist.tcp.BoboDistributedTCP method*), 104
size_stash() (*bobocep.dist.devman.BoboDeviceManager method*), 99
snapshot() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 54
snapshot() (*bobocep.cep.engine.decider.pubsub.BoboDeciderPublisher method*), 55
stash() (*bobocep.dist.devman.BoboDeviceManager method*), 99
strict (*bobocep.cep.phenom.pattern.pattern.BoboPatternBlock property*), 90
subscribe() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 54
subscribe() (*bobocep.cep.engine.decider.pubsub.BoboDeciderPublisher method*), 55
subscribe() (*bobocep.cep.engine.forwarder.forwarder.BoboForwarder method*), 63
subscribe() (*bobocep.cep.engine.forwarder.pubsub.BoboForwarderPubli method*), 64
subscribe() (*bobocep.cep.engine.producer.producer.BoboProducer method*), 65
subscribe() (*bobocep.cep.engine.producer.pubsub.BoboProducerPublish method*), 66
subscribe() (*bobocep.cep.engine.receiver.pubsub.BoboReceiverPublisher method*), 67
subscribe() (*bobocep.cep.engine.receiver.receiver.BoboReceiver method*), 69
subscribe() (*bobocep.dist.pubsub.BoboDistributedPublisher method*), 101
subscribe() (*bobocep.dist.tcp.BoboDistributedTCP method*), 104
success (*bobocep.cep.action.handler.BoboHandlerResponse attribute*), 52
success (*bobocep.cep.event.action.BoboEventAction property*), 74

T

timestamp (*bobocep.cep.event.action.BoboEventAction property*), 75
timestamp (*bobocep.cep.event.complex.BoboEventComplex property*), 76
timestamp (*bobocep.cep.event.event.BoboEvent property*), 77
timestamp (*bobocep.cep.event.simple.BoboEventSimple property*), 81
to_json_dict() (*bobocep.bobocep.BoboJSONable method*), 44
to_json_dict() (*bobocep.cep.engine.decider.runserial.BoboRunSerial method*), 59
to_json_dict() (*bobocep.cep.event.action.BoboEventAction*

method), 75
to_json_dict() (*bobocep.cep.event.complex.BoboEventComplex method*), 76
to_json_dict() (*bobocep.cep.event.event.BoboEvent method*), 78
to_json_dict() (*bobocep.cep.event.history.BoboHistory method*), 80
to_json_dict() (*bobocep.cep.event.simple.BoboEventSimple method*), 81
to_json_str() (*bobocep.bobocep.BoboJSONable method*), 44
to_json_str() (*bobocep.cep.engine.decider.runserial.BoboRunSerial method*), 59
to_json_str() (*bobocep.cep.event.action.BoboEventAction method*), 75
to_json_str() (*bobocep.cep.event.complex.BoboEventComplex method*), 76
to_json_str() (*bobocep.cep.event.event.BoboEvent method*), 78
to_json_str() (*bobocep.cep.event.history.BoboHistory method*), 80
to_json_str() (*bobocep.cep.event.simple.BoboEventSimple method*), 81

U

update() (*bobocep.cep.engine.decider.decider.BoboDecider method*), 54
update() (*bobocep.cep.engine.engine.BoboEngine method*), 61
update() (*bobocep.cep.engine.forwarder.forwarder.BoboForwarder method*), 63
update() (*bobocep.cep.engine.producer.producer.BoboProducer method*), 66
update() (*bobocep.cep.engine.receiver.receiver.BoboReceiver method*), 69
update() (*bobocep.cep.engine.task.BoboEngineTask method*), 72
urn (*bobocep.dist.device.BoboDevice property*), 97
urn (*bobocep.dist.devman.BoboDeviceManager property*), 99