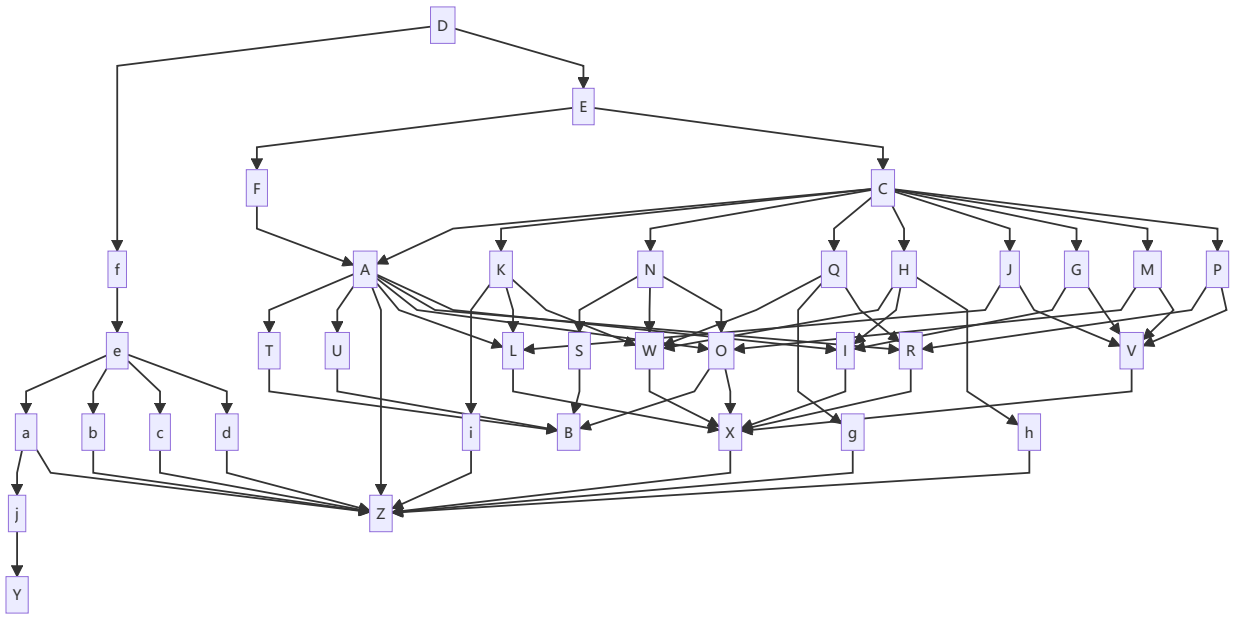
White-box Testing Document

Date	Author	Description
May 9	Zin, Aidan	The first round of white-box testing
May 10	Zin, Aidan	The second round of white-box testing

1. The structure of our codes



Node Name	Function
A	`Router.__init__(self)`
B	`Configuration.__init__(self)`
C	`Router.getResponse(self, dataInput)`
D	`Router.start(self, ip, port)`
E	`MyHttpRequestHandler.do_POST(self)`
F	`MyHttpRequestHandler.__init__(self, request, client_address, server, routerObj)`
G	`SensorCalibration.checkSuitable(self, dataInput)`
H	`SensorCalibration.getResponse(self, dataInput)`
I	`SensorCalibration.__init__(self, sensorCollectorList)`
J	`SensorStatus.checkSuitable(self, dataInput)`
K	`SensorStatus.getResponse(self, dataInput)`
L	`SensorStatus.__init__(self, sensorCollectorList)`
M	`SensorDetails.checkSuitable(self, dataInput)`
N	`SensorDetails.getResponse(self, dataInput)`
O	`SensorDetails.__init__(self, sensorCollectorList)`
P	`RealTimeData.checkSuitable(self, dataInput)`
Q	`RealTimeData.getResponse(self, dataInput)`
R	`RealTimeData.__init__(self, sensorCollectorList)`
S	`Configuration.getJsonObject(self)`
T	`Configuration.getMacAddrOfSensor(self, index)`
U	`Configuration.getNameOfSensor(self, index)`
V	`Transaction.checkSuitable(self, dataInput)`
W	`Transaction.getResponse(self, dataInput)`
X	`Transaction.__init__(self, sensorCollectorList)`
Y	`Plugin.f(data)`

Node Name	Function
Z	`SensorCollector.__init__(self, macAddr, name)`
a	`SensorCollector.__callback(self, sender, data)`
b	`SensorCollector.__connectionCheck(self)`
c	`SensorCollector.__batteryCheck(self, client)`
d	`SensorCollector.__calibrate(self, client)`
e	`SensorCollector.__start_raw(self)`
f	`SensorCollector.start(self)`
g	`SensorCollector.getRealtimeData(self)`
h	`SensorCollector.calibrate(self)`
i	`SensorCollector.getSensorStatus(self)`
j	`DataTransform.transform(self, data)`

2. Test Plan

We take an **incremental testing approach**: a **bottom-up integration approach**. In this way, the next module to be tested can be tested in combination with those modules that have been tested, and so on, adding one module at a time. This approach essentially accomplishes **unit testing** and **integration testing** at the same time. At the same time, we use the **conditional combination coverage** in logic coverage to write test cases.

According to our function call relationship diagram, it is divided into **eight layers** from bottom to top. We test all functions **from bottom to top** according to the hierarchy, and there is no requirement for testing order between functions in the same layer.

Therefore, it can be divided into **eight test sets**, each containing all white-box testing cases in that layer.

- **First Test**

  (1) `Plugin.f(data)`

- **Second Test**

  (2) `DataTransform.transform(self, data)`

  (3) `SensorCollector.__init__(self, macAddr, name)`

- **Third Test**

  (4) `SensorCollector.__callback(self, sender, data)`

  (5) `SensorCollector.__connectionCheck(self)`

(6) `SensorCollector.__batteryCheck(self, client)`

(7) `SensorCollector.__calibrate(self, client)`

(8) `SensorCollector.getSensorStatus(self)`

(9) `Configuration.__init__(self)`

(10) `Transaction.__init__(self, sensorCollectorList)`

(11) `SensorCollector.getRealtimeData(self)`

(12) `SensorCollector.calibrate(self)`

- **Forth Test**

(13) `SensorCollector.__start_raw(self)`

(14) `Configuration.getMacAddrOfSensor(self, index)`

(15) `Configuration.getNameOfSensor(self, index)`

(16) `SensorStatus.__init__(self, sensorCollectorList)`

(17) `Configuration.getJsonObject(self)`

(18) `Transaction.getResponse(self, dataInput)`

(19) `SensorDetails.__init__(self, sensorCollectorList)`

(20) `SensorCalibration.__init__(self, sensorCollectorList)`

(21) `RealTimeData.__init__(self, sensorCollectorList)`

(22) `Transaction.checkSuitable(self, dataInput)`

- **Fifth Test**

(23) `SensorCollector.start(self)`

(24) `Router.__init__(self)`

(25) `SensorStatus.getResponse(self, dataInput)`

(26) `SensorDetails.getResponse(self, dataInput)`

(27) `RealTimeData.getResponse(self, dataInput)`

(28) `SensorCalibration.getResponse(self, dataInput)`

(29) `SensorStatus.checkSuitable(self, dataInput)`

(30) `SensorCalibration.checkSuitable(self, dataInput)`

(31) `SensorDetails.checkSuitable(self, dataInput)`

(32) `RealTimeData.checkSuitable(self, dataInput)`

- **Sixth Test**

(33) `MyHttpRequestHandler.__init__(self, request, client_address, server, routerObj)`

(34) `Router.getResponse(self, dataInput)`

- **Seventh Test**

(35) `MyHttpRequestHandler.do_POST(self)`

- **Eighth Test**

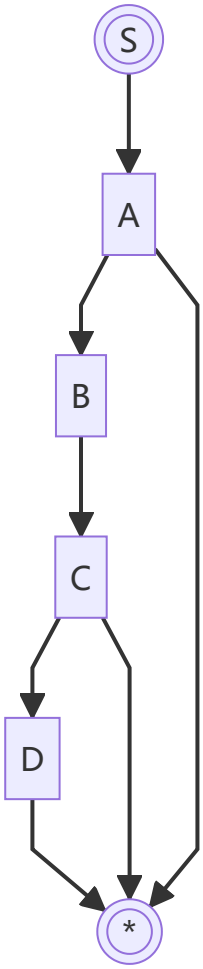
(36) `Router.start(self, ip, port)`

3. Test Cases

> **Regulations:** `S` in the figure represents the function entry, and `*` represents the function return

3.1 First Test

(1) `Plugin.f(data)`



Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	`b'x55xb8xadxe6x96x87'`	[(A, C): `(True, False)`]	SABC*	`None`
2	`b'xe4xb8xadxe6x96x87xe4xb8xadxe6x96x87xe4xb8xadxe6x96x87x96x87'`	[(A, C): `(False, True)`]	SA*	`None`
3	`b'x55x00x00x00x00x00x00x00x0x00x00x00x00x00x00x00x00x00x00'`	[(A, C): `(True, True)`]	SABCD*	`{ "X" : 0, "Y" : 0, "Z" : 0, "accX": 0 , "accY": 0 , "accZ":0, "asX" : 0 , "asY" : 0 , "asZ" : 0 }`

3.2 Second Test

(2) `DataTransform.transform(self, data)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	`b'x55x00x00x00x00x00x00x00x00x0x00x00x00x00x00x00x00x00x00x00`	No Condition	SA*	`{ "X" : 0, "Y" : 0, "Z" : 0, "accX": 0 , "accY": 0 , "accZ":0, "asX" : 0 , "asY" : 0 , "asZ" : 0 }`

(3) `SensorCollector.__init__(self, macAddr, name)`

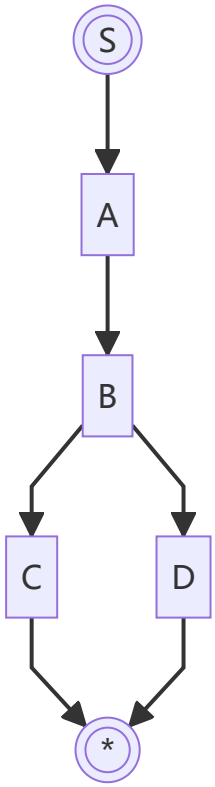
Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	macAddr = `"F2:02:E0:8D:B8:05"`, name = `"R1"`	no condition	SA*	macAddr = `"F2:02:E0:8D:B8:05"`, name = `"R1"`, cache = `None`, type(cacheTime) = `datetime.datetime`, needCalibrate = `False`, type(lastCalibrate) = `datetime.datetime`, connected = `False`, battery = `0`

3.3 Third Test

(4) `SensorCollector.__callback(self, sender, data)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	sender = `None` , data = `b'x55x00x00x00x00x00x00x00x0x00x00x00x00x00x00x00x00x00x00`	No Conditions	SA*	cache = { "X" : 0, "Y" : 0, "Z" : 0, "accX": 0 , "accY": 0 , "accZ":0, "asX" : 0 , "asY" : 0 , "asZ" : 0 }, type(cacheTime) = `datetime.datetime`

(5) `SensorCollector.__connectionCheck(self)`



Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	`delta.total_seconds()` > `TIME_OUT_SPAN`	[B:`True`]	SABC*	`False`
2	`delta.total_seconds()` <= `TIME_OUT_SPAN`	[B:`False`]	SABD*	`True`

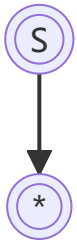
(6) `SensorCollector.__batteryCheck(self, client)`

```
def __batteryCheck(self, client: BleakClient) -> int:  # TODO: read battery
    # print(self.cache)
    return 100
```

> Obviously, there's no problem with the code.

(7) `SensorCollector.__calibrate(self, client)`

```
def __calibrate(self, client) -> None:  # TODO: calibrate
    self.needCalibrate = False
```

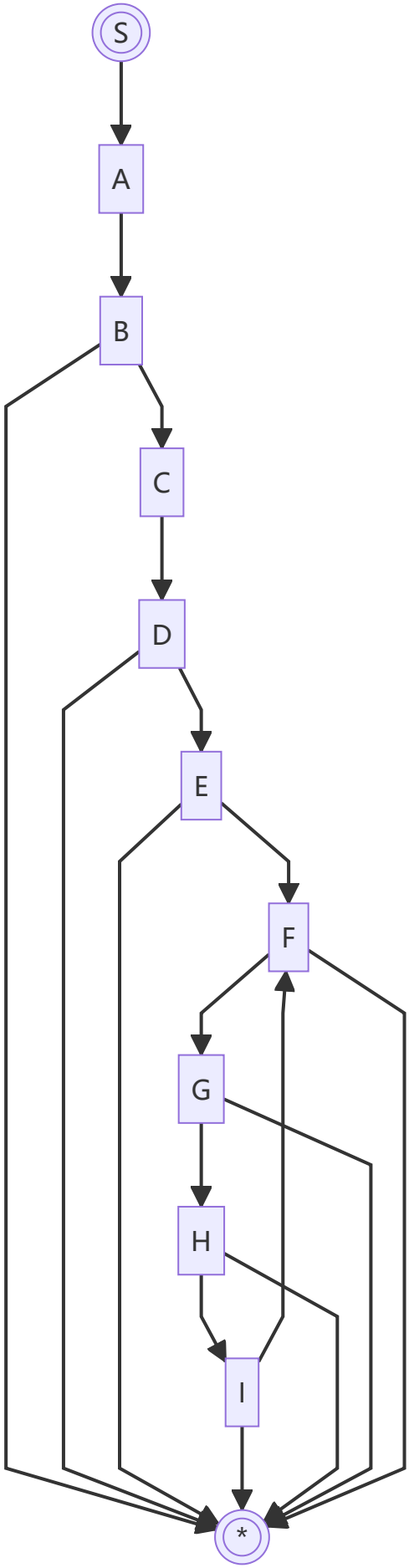


> Obviously, there's no problem with the code.

(8) `SensorCollector.getSensorStatus(self)`

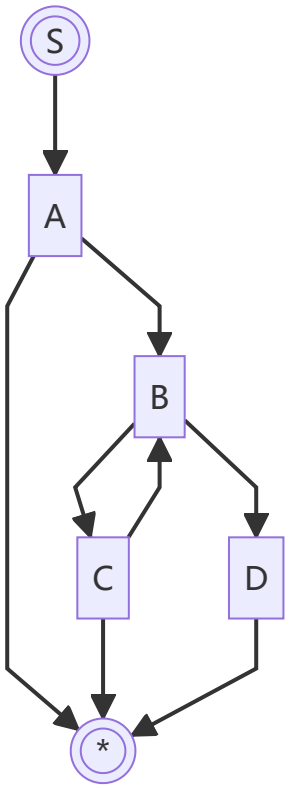
Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	`connected = False; battery = 0;`	no condition	SA*	`{"connect": False, "battery": 0}`

(9) `Configuration.__init__(self)`


S
A
B
C
D
E
F
G
H
I
*

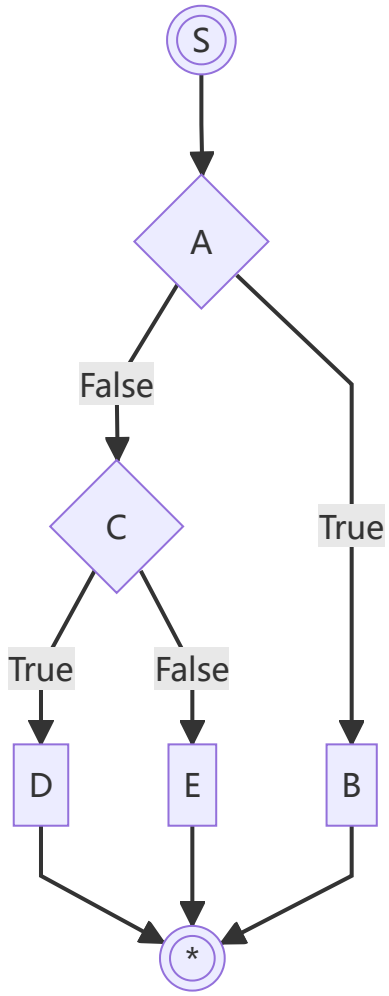
Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	config.json : 无	[B : `False`]	SAB*	None
2	config.json : 999	[(B, D) : (`True ,False`]	SABCD*	None
3	config.json : [{},{},{},{},{},{},{}]	[(B,D, E): `(True, True, False)`]	SABCDE*	None
4	config.json : [[1, 4, 6]]	[(B,D,E,G): `(True, True, False, False)`]	SABCDEFG	None
5	config.json : [{"name": "R1", "macAddr": "F2:02:E0:8D:B8:05"}]	[(B,D,E,G): `(True, True, True, True)`]	SABCDEFGHIF*	None

(10) `Transaction.__init__(self, sensorCollectorList)`



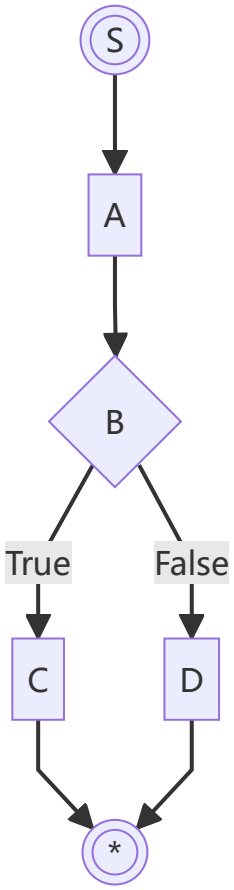
Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	sensorCollectorList = `666`	[A : `False`]	SA*	sensorCollectorList = `None`
2	sensorCollectorList = `[SensorCollector("F2:02:E0:8D:B8:05", "R1")]`	[(A, B1, B2, C) : `(True, True, False), True`]	SABCBD*	sensorCollectorList = `[SensorCollector("F2:02:E0:8D:B8:05", "R1")]`
3	`sensorCollectorList = [2]`	[(A, C) : `(True, False)`]	SABC*	sensorCollectorList = `None`

(11) `SensorCollector.getRealtimeData(self)`



Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	connected = `True`; cache = `{"X":1, "Y":1, "Z":1, "accX":1, "accY":1, "accZ":1, "asX":1, "asY":1, "asZ":1}`	`connected, cache is not None`	SACD*	`{"X":1, "Y":1, "Z":1, "accX":1, "accY":1, "accZ":1, "asX":1, "asY":1, "asZ":1}`
2	connected = `True`; cache = `None`	`connected, cache is None`	SACE*	`INVALID_DATA`
3	connected = `False`	`not connected`	SAB*	`INVALID_DATA`

(12) `SensorCollector.calibrate(self)`



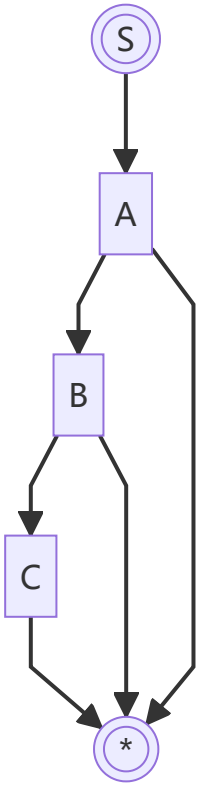
Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	lastCalibrate = `datetime.datetime.utcfromtimestamp(0)`	`deltaTime.total_seconds() > CALIBRATE_SPAN`	SABC*	`type(lastCalibrate) = datetime.datetime; needCalibrate = True; returnValue = True;`
2	lastCalibrate = `datetime.datetime.utcnow()`	`deltaTime.total_seconds() <= CALIBRATE_SPAN`	SABD*	`type(lastCalibrate) = datetime.datetime; returnValue = False;`

3.4 Forth Test

(13) `SensorCollector.__start_raw(self)`

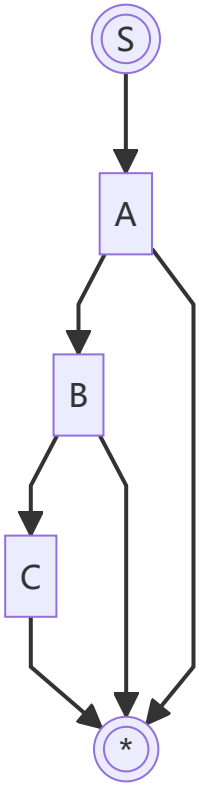
In the testing, we used `pytest` and `MagicMock` to simulate the `BleakClient` class, and `AsyncMock` to simulate some methods of the `BleakClient` class, such as `connect`, `start_ notify`、`write_ gatt_char` and so on. Then, we called the `__ start_raw (client)` method simulates the process of interaction between the `BleakClient` object and the incoming parameter `client`.
Finally, we use various `assert` statements to check whether the simulated method is correctly called to ensure the correctness and stability of the method.

(14) `Configuration.getMacAddrOfSensor(self, index)`



Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	-1	[(A, B): `(False, True)`]	SA*	`None`
2	6	[(A, B): `(True, False)`]	SAB*	`None`
3	0	[(A, B): `(True, True)`]	SABC*	`"F2:02:E0:8D:B8:05"`

(15) `Configuration.getNameOfSensor(self, index)`



Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	-1	[(A, B): `(False, True)`]	SA*	`None`
2	6	[(A, B): `(True, False)`]	SAB*	`None`
3	0	[(A, B): `(True, True)`]	SABC*	`"R1"`

(16) `SensorStatus.__init__(self, sensorCollectorList)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	sensorCollectorList = `[SensorCollector("F2:02:E0:8D:B8:05", "R1")]`	no condition	SA*	sensorCollectorList = `[SensorCollector("F2:02:E0:8D:B8:05", "R1")]`

(17) `Configuration.getJsonObject(self)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	data = `[ {"name": "R1", "macAddr": "F2:02:E0:8D:B8:05"},{"name": "R2", "macAddr": "C4:39:0D:A9:91:89"}, {"name": "R3", "macAddr": "E8:67:FE:A6:D4:3C", {"name": "L1", "macAddr": "D1:7A:2A:54:02:95"}, {"name": "L2", "macAddr": "D7:0F:4F:1D:4F:B5"}, {"name": "L3", "macAddr": "E6:7A:B7:B0:45:9D"}]`	no condition	SA*	`[ {"name": "R1", "macAddr": "F2:02:E0:8D:B8:05"}, {"name": "R2", "macAddr": "C4:39:0D:A9:91:89"}, {"name": "R3", "macAddr": "E8:67:FE:A6:D4:3C", {"name": "L1", "macAddr": "D1:7A:2A:54:02:95"}, {"name": "L2", "macAddr": "D7:0F:4F:1D:4F:B5"}, {"name": "L3", "macAddr": "E6:7A:B7:B0:45:9D"}]`

(18) `Transaction.getResponse(self, dataInput)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{"name": "test", "macAddr": "test"}`	no condition	SA*	`ERROR_MESSAGE`

(19) `SensorDetails.__init__(self, sensorCollectorList)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	sensorCollectorList = `[SensorCollector("F2:02:E0:8D:B8:05", "R1")]`, config = `Configuration()`	no condition	SA*	sensorCollectorList = `[SensorCollector("F2:02:E0:8D:B8:05", "R1")]`, self.config = `Configuration()`

(20) `SensorCalibration.__init__(self, sensorCollectorList)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	sensorCollectorList = `[SensorCollector("F2:02:E0:8D:B8:05", "R1")]`	no condition	SA*	sensorCollectorList = `[SensorCollector("F2:02:E0:8D:B8:05", "R1")]`

(21) `RealTimeData.__init__(self, sensorCollectorList)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	sensorCollectorList = `[SensorCollector("F2:02:E0:8D:B8:05", "R1")]`	no condition	SA*	sensorCollectorList = `[SensorCollector("F2:02:E0:8D:B8:05", "R1")]`

(22) `Transaction.checkSuitable(self, dataInput)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{"name": "test", "macAddr": "test"}`	no condition	SA*	`False`

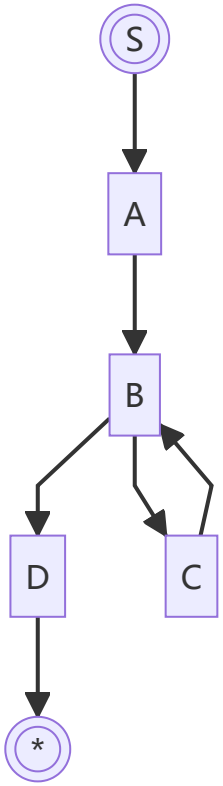
3.5 Fifth Test

(23) `SensorCollector.start(self)`

```
1 def start(self):
2     self.thread = threading.Thread(target=lambda:
asyncio.run(self.__start_raw()))
3     self.thread.start()
```

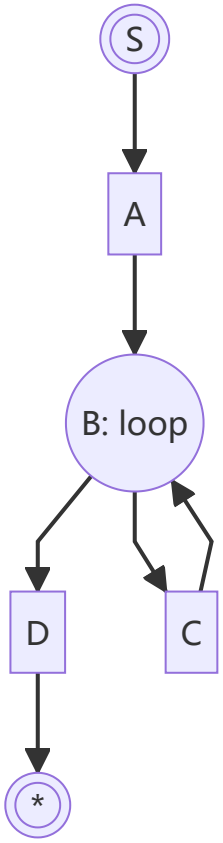
Obviously, the correctness of this function itself depends on the correctness of function `SensorCollector.__start__raw(self)`.

(24) `Router.__init__(self)`



Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	`None`	No Conditions	SABCBCBCBCBCBD*	`[RealTimeData(self.sensorCollectorList), SensorDetails(self.sensorCollectorList, self.config), SensorStatus(self.sensorCollectorList), SensorCalibration(self.sensorCollectorList)]`

(25) `SensorStatus.getResponse(self, dataInput)`

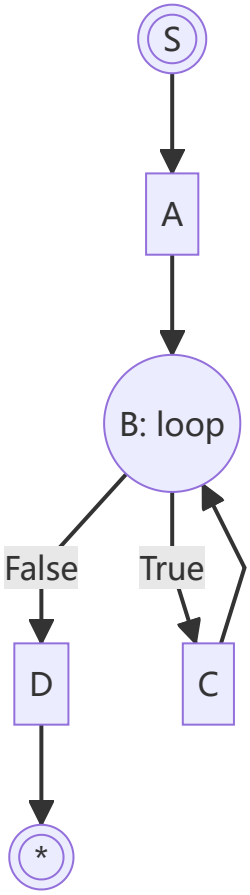


Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{"type":"GetSensorStatus"}`	no condition	SABC..BD*	`{'0': {'connect': False, 'battery': 0},'1': {'connect': False, 'battery': 0},'2': {'battery': 0, 'connect': False},'3': {'battery': 0, 'connect': False},'4': {'battery': 0, 'connect': False},'5': {'battery': 0, 'connect': False}}`

(26) `SensorDetails.getResponse(self, dataInput)`

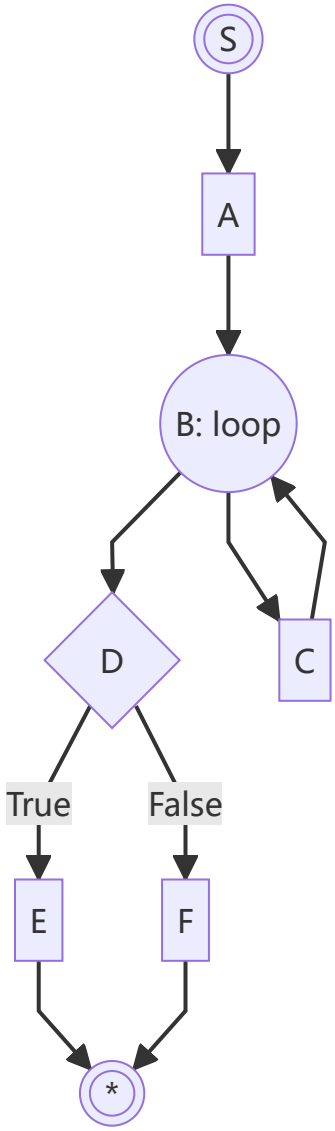
Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{ "type":"GetSensorDetails" }`	no condition	SA*	`[{"name": "R1","macAddr": "F2:02:E0:8D:B8:05"},{"name": "R2","macAddr": "C4:39:0D:A9:91:89"},{"name": "R3","macAddr": "E8:67:FE:A6:D4:3C"},{"name": "L1","macAddr": "D1:7A:2A:54:02:95"},{"name": "L2","macAddr": "D7:0F:4F:1D:4F:B5"},{"name": "L3","macAddr": "E6:7A:B7:B0:45:9D"}]`

(27) `RealTimeData.getResponse(self, dataInput)`



Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{"type":"GetRealtimeData"}`	no condition	SABC..BD*	`{"R1": INVALID_DATA, "R2": INVALID_DATA, "R3": INVALID_DATA, "L1": INVALID_DATA, "L2": INVALID_DATA, "L3": INVALID_DATA, 'timestamp': (a float)}`

(28) `SensorCalibration.getResponse(self, dataInput)`



Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{"type":"SensorCalibration"}` & 6 sensors have been calibrated for `more than 10 seconds` since the last calibration	`ans is True`	SABC..BDE*	`{"type": "CalibrationSuccess"}`
2	dataInput = `{"type":"SensorCalibration"}` & Six sensors were just calibrated `within 10 seconds`	`ans is False`	SABC..BDF*	`{"type": "CalibrationFailure"}`

(29) `SensorStatus.checkSuitable(self, dataInput)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{"type": "GetSensorStatus"}`	no condition	SA*	`True`

(30) `SensorCalibration.checkSuitable(self, dataInput)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{"type": "SensorCalibration"}`	no condition	SA*	`True`

(31) `SensorDetails.checkSuitable(self, dataInput)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{"type": "GetSensorDetails"}`	no condition	SA*	`True`

(32) `RealTimeData.checkSuitable(self, dataInput)`

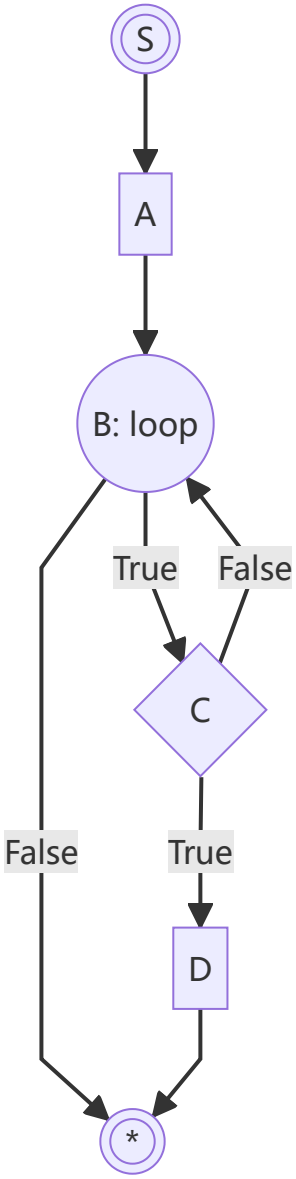
Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{"type": "GetRealtimeData"}`	no condition	SA*	`True`

3.6 Sixth Test

(33) `MyHttpRequestHandler.__init__(self, request, client_address, server, routerObj)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	routerObj = Router()	No Condition	SA*	router = Router()

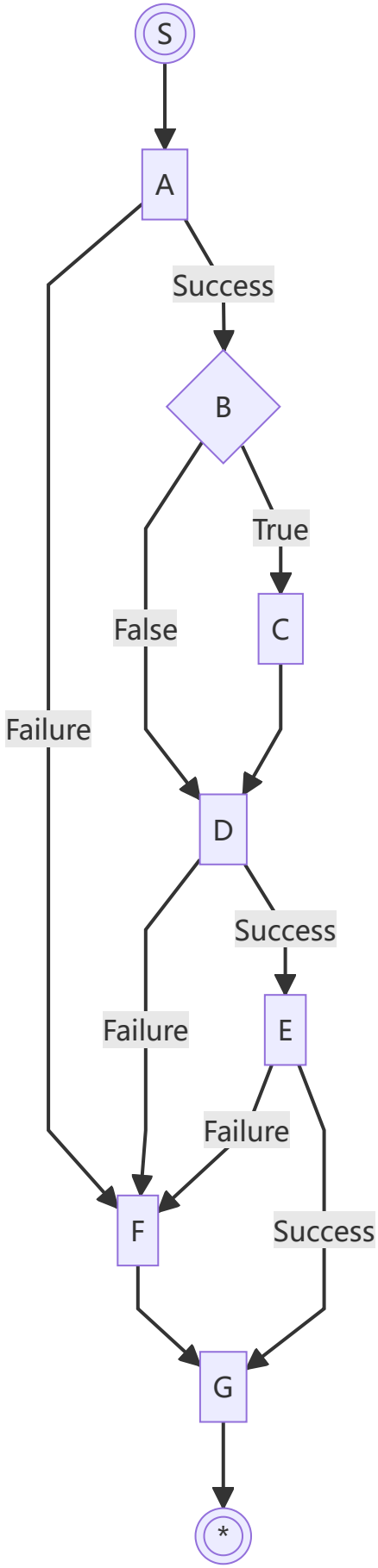
(34) `Router.getResponse(self, dataInput)`



Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	dataInput = `{"type": "GetSensorDetails"}`	`transcation.checkSuitable(dataInput) = True`	SABC..BCD*	`[{"name": "R1","macAddr": "F2:02:E0:8D:B8:05"},{"name": "R2","macAddr": "C4:39:0D:A9:91:89"},{"name": "R3","macAddr": "E8:67:FE:A6:D4:3C"},{"name": "L1","macAddr": "D1:7A:2A:54:02:95"},{"name": "L2","macAddr": "D7:0F:4F:1D:4F:B5"},{"name": "L3","macAddr": "E6:7A:B7:B0:45:9D"}]`
2	dataInput = `{"type": "Test"}`	`transcation.checkSuitable(dataInput) = False`	SABCB*	`{'type': 'TypeError'}`

3.7 Seventh Test

(35) `MyHttpRequestHandler.do_POST(self)`


S
A
Success
B
True
False
C
Failure
D
Success
Failure
E
Failure
F
Success
G
*

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	client request: `clientRequest({"type": "GetSensorStatus"}, "127.0.0.1", "40096")`	`A: Success, B: True, D: Success, E: Success`	SABCDEG*	`response = {'0': {'connect': False, 'battery': 0}, '1': {'connect': False, 'battery': 0}, '2': {'connect': False, 'battery': 0}, '3': {'connect': False, 'battery': 0}, '4': {'connect': False, 'battery': 0}, '5': {'connect': False, 'battery': 0}}`
2	client request: `clientRequest({"name":"Test"}, "127.0.0.1", "40096")`	`A: Success, B: True, D: Success, E: Failure`	SABCDEFG*	`ERROR_MESSAGE`

3.8 Eighth Test

(36) `Router.start(self, ip, port)`

Case Number	Test Case	Coverage Conditions	Overlay Path	Expected Results
1	ip = `"127.0.0.1"`, port = `40096`	No condition	SA*	"Server started on http://127.0.0.1:40096"

4. Description of automated testing tools

We used `Python`'s third-party library, `Pytest`, to complete automated white-box testing. `Pytest` is a relatively mature and fully functional Python testing framework. It provides comprehensive online documentation, with a large number of third-party plugins and built-in help, suitable for many small or large projects. `Pytest` is flexible and easy to learn, and can capture standard output during print debugging and test execution, making it suitable for simple unit testing to complex functional testing. You can also execute `nose`, `unittest`, and `doctest` style test cases, and even `Django` and `trial`. Support good integration practices, support extended `xUnit` style setups, and support `non Python testing`. Support for generating test coverage reports and supporting `PEP8` compatible encoding styles.
The main features are as follows:

- Simple and flexible, easy to learn, with rich documentation;
- Support parameterization, allowing fine-grained control of the test cases to be tested;
- It can support simple unit testing and complex functional testing, and can also be used for automation testing such as `selenium/appnium` and interface automation testing (`pytest` + `requests`);
- `Pytest` has many third-party plugins and can be customized and extended, such as `pytest-selenium` (integrated selenium), `pytest-html` (perfect HTML test report generation), `pytest-rerunfailures` (repeated execution of failed cases), `pytest-xdist` (multi CPU distribution), etc;

  > In our testing, we used `pytest-html` to generate a test report: `report.html`.

- Skip and xfail handling of test cases;
- Can be well integrated with CI tools, such as `Jenkins`
- The report framework - alloure also supports `pytest`

5. Test Report

5.1 The first round of white-box testing

A total of 52 test items were tested on 36 functions, of which 51 test items of 34 functions passed and 1 test item of 1 function failed.

The details are shown in the table below.

Id	Function	Test Result	Review
(1)	`Router.__init__(self)`	**Passed**	
(2)	`Configuration.__init__(self)`	**Passed**	
(3)	`Router.getResponse(self, dataInput)`	**Passed**	
(4)	`Router.start(self, ip, port)`	**Passed**	
(5)	`MyHttpRequestHandler.do_POST(self)`	**Passed**	

Id	Function	Test Result	Review
(6)	`MyHttpRequestHandler.__init__(self, request, client_address, server, routerObj)`	**Passed**	
(7)	`SensorCalibration.checkSuitable(self, dataInput)`	**Passed**	
(8)	`SensorCalibration.getResponse(self, dataInput)`	**Failed on Test Case 2**	**Analysis:** *the return value `{"type": " CalibrationFailure"}` had **an additional space** before the `CalibrationFailure`* **Modification:** `{"type": " CalibrationFailure"}` -> `{"type": "CalibrationFailure"}`
(9)	`SensorCalibration.__init__(self, sensorCollectorList)`	**Passed**	
(10)	`SensorStatus.checkSuitable(self, dataInput)`	**Passed**	
(11)	`SensorStatus.getResponse(self, dataInput)`	**Passed**	
(12)	`SensorStatus.__init__(self, sensorCollectorList)`	**Passed**	
(13)	`SensorDetails.checkSuitable(self, dataInput)`	**Passed**	
(14)	`SensorDetails.getResponse(self, dataInput)`	**Passed**	
(15)	`SensorDetails.__init__(self, sensorCollectorList)`	**Passed**	
(16)	`RealTimeData.checkSuitable(self, dataInput)`	**Passed**	
(17)	`RealTimeData.getResponse(self, dataInput)`	**Passed**	
(18)	`RealTimeData.__init__(self, sensorCollectorList)`	**Passed**	
(19)	`Configuration.getJsonObject(self)`	**Passed**	
(20)	`Configuration.getMacAddrOfSensor(self, index)`	**Passed**	
(21)	`Configuration.getNameOfSensor(self, index)`	**Passed**	

Id	Function	Test Result	Review
(22)	`Transaction.checkSuitable(self, dataInput)`	**Passed**	
(23)	`Transaction.getResponse(self, dataInput)`	**Passed**	
(24)	`Transaction.__init__(self, sensorCollectorList)`	**Passed**	
(25)	`Plugin.f(data)`	**Passed**	
(26)	`SensorCollector.__init__(self, macAddr, name)`	**Passed**	
(27)	`SensorCollector.__callback(self, sender, data)`	**Passed**	
(28)	`SensorCollector.__connectionCheck(self)`	**Passed**	
(29)	`SensorCollector.__batteryCheck(self, client)`	**Passed**	
(30)	`SensorCollector.__calibrate(self, client)`	**Passed**	
(31)	`SensorCollector.__start_raw(self)`	**Passed**	
(32)	`SensorCollector.start(self)`	**Passed**	
(33)	`SensorCollector.getRealtimeData(self)`	**Passed**	
(34)	`SensorCollector.calibrate(self)`	**Passed**	
(35)	`SensorCollector.getSensorStatus(self)`	**Passed**	
(36)	`DataTransform.transform(self, data)`	**Passed**	

5.2 The second round of white-box testing

After modifying `SensorCalibration.getResponse(self, dataInput)`, all 52 test items of 36 functions passed the test. The test results are as follows, and the detailed test report is in `report.html`.

```
PS D:\OneDrive\Desktop\WhiteBox_Testing\code> pytest --verbose
============================================= test session starts =============================================
platform win32 -- Python 3.8.3, pytest-7.1.2, pluggy-1.0.0 -- D:\LenovoSoftstore\Install\Anaconda3\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.8.3', 'Platform': 'Windows-10-10.0.22621-SP0', 'Packages': {'pytest': '7.1.2', 'pluggy': '1.0.0'
}, 'Plugins': {'anyio': '3.6.2', 'asyncio': '0.21.0', 'html': '3.2.0', 'metadata': '2.0.4'}}
rootdir: D:\OneDrive\Desktop\WhiteBox_Testing\code
plugins: anyio-3.6.2, asyncio-0.21.0, html-3.2.0, metadata-2.0.4
asyncio: mode=strict
collected 52 items

test_1.py::test_f[U\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00-expected_val0] PASSED [
 1%]
test_2.py::test_transform[U\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00-expected_val0] P
ASSED [  3%]
test_2.py::test_init[F2:02:E0:8D:B8:05-R1] PASSED                                                    [  5%]
test_2.py::test_init[C4:39:0D:A9:91:89-R2] PASSED                                                    [  7%]
test_3.py::TestSensorCollector::test_callback[U\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00-expected_val0] PASSED [  9%]
test_3.py::test_connectionCheck[0-True] PASSED                                                       [ 11%]
test_3.py::test_connectionCheck[3-False] PASSED                                                      [ 13%]
test_3.py::test_getSensorStatus[F2:02:E0:8D:B8:05-R1-expected_val0] PASSED                           [ 15%]
test_3.py::test_ConfigurationInit PASSED                                                             [ 17%]
test_3.py::test_callback[data0] PASSED                                                               [ 19%]
test_3.py::test_getRealtimeData[True-cache0-expected_val0] PASSED                                    [ 21%]
test_3.py::test_getRealtimeData[True-None-expected_val1] PASSED                                      [ 23%]
test_3.py::test_getRealtimeData[False-None-expected_val2] PASSED                                     [ 25%]
test_3.py::test_calibrate[lastCalibrate0-True-True-datetime] PASSED                                  [ 26%]
test_3.py::test_calibrate[lastCalibrate1-False-False-datetime] PASSED                                [ 28%]
test_4.py::test_my_function PASSED                                                                   [ 30%]
test_4.py::test_getMacAddrOfSensor[0-F2:02:E0:8D:B8:05] PASSED                                       [ 32%]
test_4.py::test_getMacAddrOfSensor[1-C4:39:0D:A9:91:89] PASSED                                       [ 34%]
test_4.py::test_getNameOfSensor[0-R1] PASSED                                                         [ 36%]
test_4.py::test_getNameOfSensor[1-R2] PASSED                                                         [ 38%]
test_4.py::test_SensorStatusInit[data0] PASSED                                                       [ 40%]
test_4.py::test_getJsonObject[data0] PASSED                                                          [ 42%]
test_4.py::test_TransactionGetResponse PASSED                                                        [ 44%]
test_4.py::test_SensorDetailsInit[data0] PASSED                                                      [ 46%]
test_4.py::test_SensorCalibrationInit[data0] PASSED                                                  [ 48%]
test_4.py::test_RealTimeDataInit[data0] PASSED                                                       [ 50%]
test_4.py::test_TransactionCheckSuitable PASSED                                                      [ 51%]
test_5.py::test_SensorCollectorStart PASSED                                                          [ 53%]
test_5.py::test_init PASSED                                                                          [ 55%]
test_5.py::test_SensorStatusGetResponse[dataInput0-expected_val0] PASSED                             [ 57%]
test_5.py::test_SensorDetailsGetResponse[data0] PASSED                                               [ 59%]
test_5.py::test_RealTimeDataGetResponse[input_val0-expected_val0-names0] PASSED                      [ 61%]
test_5.py::test_SensorCalibrationGetResponse[input_val0-lastCalibrate0-expected_val0] PASSED         [ 63%]
test_5.py::test_SensorCalibrationGetResponse[input_val1-lastCalibrate1-expected_val1] PASSED         [ 65%]
test_5.py::test_SensorStatusCheckSuitable PASSED                                                     [ 67%]
test_5.py::test_SensorCalibrationCheckSuitable PASSED                                                [ 69%]
test_5.py::test_SensorDetailsCheckSuitable PASSED                                                    [ 71%]
test_5.py::test_RealTimeDataCheckSuitable PASSED                                                     [ 73%]
test_6.py::test_MyHttpRequestHandlerInit[dataInput0-expected_val0] PASSED                            [ 75%]
test_6.py::test_MyHttpRequestHandlerInit[dataInput1-expected_val1] PASSED                            [ 76%]
test_6.py::test_MyHttpRequestHandlerInit[dataInput2-expected_val2] PASSED                            [ 78%]
test_6.py::test_MyHttpRequestHandlerInit[dataInput3-expected_val3] PASSED                            [ 80%]
test_6.py::test_getResponse[dataInput0-expected_val0] PASSED                                         [ 82%]
test_6.py::test_getResponse[dataInput1-expected_val1] PASSED                                         [ 84%]
test_7.py::test_do_POST[dataInput0-expected_val0] PASSED                                             [ 86%]
test_7.py::test_do_POST[dataInput1-expected_val1] PASSED                                             [ 88%]
test_7.py::test_do_POST[dataInput2-expected_val2] PASSED                                             [ 90%]
test_7.py::test_do_POST[dataInput3-expected_val3] PASSED                                             [ 92%]
test_8.py::test_do_POST[dataInput0-expected_val0] PASSED                                             [ 94%]
test_8.py::test_do_POST[dataInput1-expected_val1] PASSED                                             [ 96%]
test_8.py::test_do_POST[dataInput2-expected_val2] PASSED                                             [ 98%]
test_8.py::test_do_POST[dataInput3-expected_val3] PASSED                                             [100%]

============================================= 52 passed in 30.35s =============================================
PS D:\OneDrive\Desktop\WhiteBox_Testing\code>
```

6. Appendix

Name	Value
`ERROR_MESSAGE`	`{"type":"TypeError"}`
`CALIBRATE_SPAN`	`10`
`INVALID_DATA`	`{"X" : 0, "Y" : 0, "Z" : 0, "accX": 0, "accY": 0, "accZ": 0, "asX" : 0, "asY" : 0, "asZ" : 0}`

Name	Value
`TIME_OUT_SPAN`	1