

# Instructions for test\_fault.py

Abdullah Alsafar

August 2025

## 1 Notes

1. The **test\_fault.py** file can be found here:  
<https://github.com/AbdullahAGit/microgrid-fault-test>
2. It is meant to be used with the Typhoon HIL Schematic named  
**"microgrid\_Data generation.tse"**
3. Please feel free to contact me at [abdullah.alsafar@torontomu.ca](mailto:abdullah.alsafar@torontomu.ca) if you have any questions

## 2 Introduction

Typhoon HIL provides an testing software "TyphoonTest" which can be used in conjunction with Typhoon schematic files. TyphoonTest uses a mix of the pytest framework as well as their own APIs which has documentation and can be found at:

- [https://www.typhoon-hil.com/documentation/typhoon-hil-api-documentation/typhoon\\_api.html](https://www.typhoon-hil.com/documentation/typhoon-hil-api-documentation/typhoon_api.html)
- <https://www.typhoon-hil.com/documentation/typhoon-hil-typhoontest-library/>

These links are also available in the "Help" section on the Typhoon HIL Control Center.

This document will serve as an explanation for some beginner pytest features such as scopes, fixtures and test cases as well as go in to detail about how the **test\_fault.py** file works. For more information on Typhoon HIL and TyphoonTest, the learning modules provided by HIL Academy are an excellent source.

## 3 Pytest Basics

The **test\_fault.py** file makes use of 2 simple pytest features: test cases and fixtures.

- Test cases are usually parameterized as dictate what happens while the test is occuring. In the TyphoonTest IDE, the runtime of tests is shown under the "Test" Section.
- Fixtures on the other hand are commonly used to setup and teardown i.e before a test starts and after a test finishes. This, in the context of Typhoon HIL, can include things such as loading the model, starting the simulation, setting fixed variables (that is, they won't change in the middle of a test) and stopping the simulation.

Fixtures contain a scope which can define when you want them to run. For example, a fixture with the *'function'* scope occurs for each and every test case. However, a *'module'* scope occurs only once for a module which usually means that the fixture setup occurs before the first test and the fixture teardown occurs after the last test.

Fixtures are activated upon being called by other tests in their parameters. In fact, fixtures can even be by called other fixtures in their parameters, which is useful because you can "chain" fixtures together, as in:

- The test calls a fixture that runs for every test with the *function* scope, and that fixture calls another fixture that once for all tests with the *module* scope.

## 4 test\_fault.py

### 4.1 load\_model Fixture

For test\_fault.py, there are 4 fixtures and 1 test case. All 4 fixtures have the *'function'* scope and so they occur for each and every test. The **load\_model** fixture is responsible for:

1. Loading the schematic model .tse file
2. Setting the fault\_resistance and fault\_type property values on the enabled faults
  - Some fault types require a connection to GND and some others do not. So, the fixture is also responsible for connecting the enabled faults to GND if required, which means creating a "GND" object near the position of the fault in the schematic. This is what these lines are for:

```

67 if "GND" in set_fault_type:
68     fault_gnd = mdl.create_component(
69         'core/Ground',
70         name='fault_gnd',
71         position = mdl.get_position(faultHandle)
72     )
73     mdl.create_connection(mdl.term(faultHandle, 'GND'), mdl.term(fault_gnd, 'node'))

```

3. Saving the changes and compiling the model
4. Loading the compiled .cpd model
5. Setting all fixed inputs such as:
  - The operation mode of the Diesel Genset and Battery ESS
  - The average, max and min wind speed for the Wind Turbine
  - The grid vrms\_cmd and grid freq\_cmd to 1
  - The rates of change for all subsystems
  - Enabling all subsystems and microgrid controller
6. And finally, starting the simulation and, during fixture teardown, stopping the simulation

All of the steps listed occur during the "setup" stage of the fixture, except for stopping the simulation, which occurs during the "teardown" stage of the fixture which can be seen here:

```

160     hil.start_simulation()
161
162     yield
163     #Fixture Teardown
164
165     hil.stop_simulation()

```

As a note, the *yield* keyword determines the boundary between the setup and teardown of a fixture. However, it can also return various variables which are helpful when it is called by other fixtures and tests, which will be a vital detail later.

## 4.2 test\_faults Test Case

The test\_faults test case has 3 parameters:

- **set\_fault\_resistance**
- **set\_fault\_type**
- **set\_fault**

Other values can be added to **set\_fault\_resistance** but for **set\_fault\_type** the fault type must be available in the drop-down menu for faults in the schematic. For **set\_fault** the fault name must be an existing fault in the schematic.

For these parameters, every unique combination is run by **test\_faults**. The order is decided by the order they are called by the **test\_faults** function:

```

202 def test_faults(load_model, set_fault_resistance, set_fault_type, set_fault):

```

According to the order, all resistances will be tested for each fault type, and all fault types will be tested for each fault that has been enabled. Therefore, while testing, the next fault will only begin when all other combinations for the test before that have ran.

At the beginning of the test, there are many print statements. These are present just for testing purposes and ensuring the desired output is achieved. Because the fixture setup has already ran by this point, the simulation is on.

Before beginning capturing all desired signals, the simulation waits 20 seconds. This is important because the specific model **microgrid\_Data generation.tse** requires a little less than 20 seconds to reach steady state. After the 20 seconds are finished, the capture begins.

```

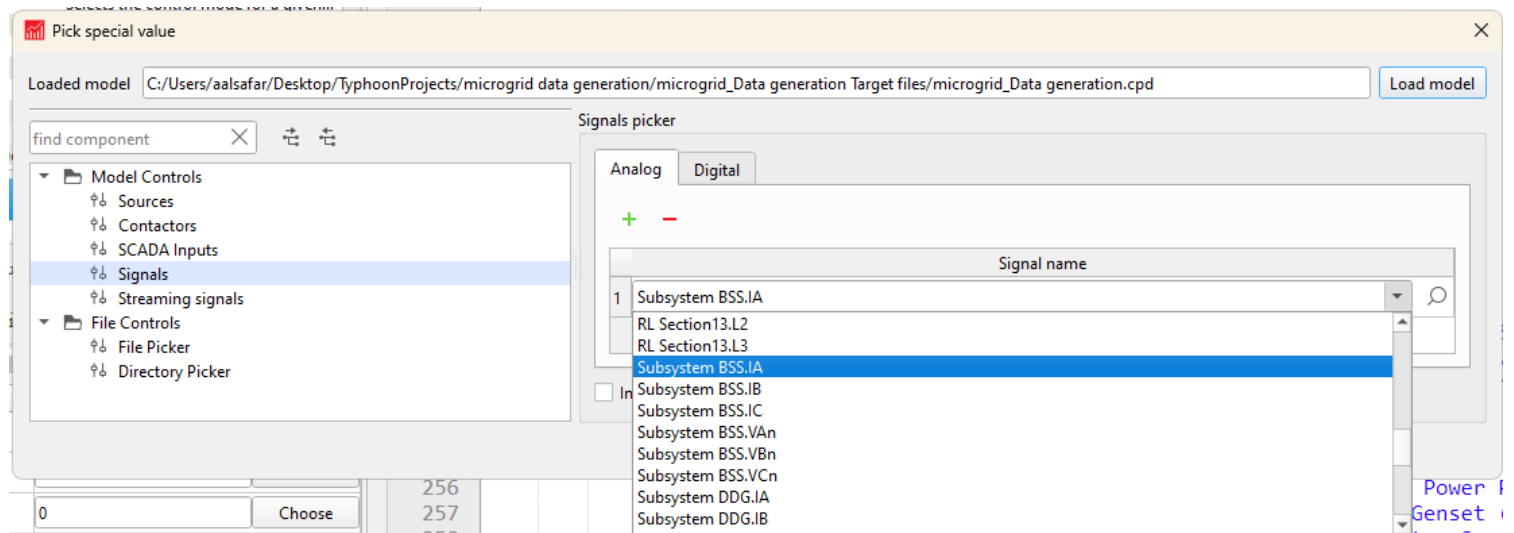
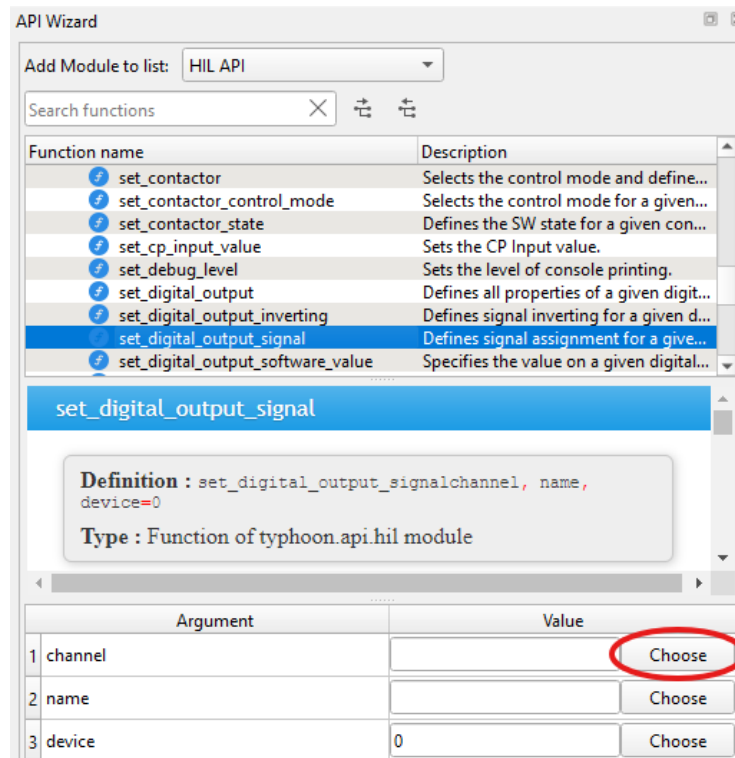
241 cap_duration = 1
242 time_before_fault = cap_duration/2
243 cap.start_capture(duration=cap_duration,
244                 rate=fs,
245                 signals=[
246
247                     'Grid1.Vc', 'Grid1.Vb', 'Grid1.Va',
248                     'PCC_monitor.S1_fb',
249                     'Grid1.Ic', 'Grid1.Ib', 'Grid1.Ia',
250                     'Subsystem BSS.IA', 'Subsystem BSS.IB', 'Subsystem BSS.IC',
251                     'Subsystem WT-B.IA', 'Subsystem WT-B.IB', 'Subsystem WT-B.IC',
252                     'Subsystem WT-B.VAn', 'Subsystem WT-B.VBn', 'Subsystem WT-B.VCn',
253                     'PCC_monitor.Va', 'PCC_monitor.Vb', 'PCC_monitor.Vc',
254                     'PCC_monitor.VA', 'PCC_monitor.VB', 'PCC_monitor.VC',
255                     'Grid UI1.Vrms_meas_kV', 'Grid UI1.Qmeas_kVAR', 'Grid UI1.Pmeas_kW',
256                     'Wind Power Plant (Generic) UI1.Pmeas_kW', 'PV Power Plant (Generic) UI1.Pmeas_kW',
257                     'Battery ESS (Generic) UI1.Pmeas_kW', 'Diesel Genset (Generic) UI1.Pmeas_kW',
258                     'PCC_monitor.Synch_check.PLLs.VABC', 'PCC_monitor.Synch_check.PLLs.Vabc'
259                     #'Wind Power Plant (Generic) UI1.wind_speed_m_per_s',
260                     #'Wind Power Plant (Generic) UI1.MCB_status', 'PV Power Plant (Generic) UI1.MCB_status',
261                     #'Diesel Genset (Generic) UI1.MCB_status', 'Battery ESS (Generic) UI1.MCB_status',
262                 ],)

```

- The *cap\_duration* variable defines how long the capture lasts for in seconds. In this case, it is 1 second.
- The *time\_before\_fault* variable defines when the selected fault is enabled. In this case, it is halfway after the fault begins and so the fault occurs at 0.5 seconds.
- The *rate* defines the frequency at which the selected signals are sampled. The variable *fs* was set at the beginning of the script to be 100e3 and so the signals are sampled every 1/100e3 or 0.00001 seconds. Increase or decrease this number to change the resolution of the capture.
- The *signals* list defines all signals that are to be captured. The signals in the list must have the correct schematic names otherwise the capture returns an error.

Note:

- An easy way to find the correct name for signals is by using the API Wizard's "Choose" button on a value after selecting any API function. The API Wizard is also helpful for inserting other API functions like *set\_contactor* and *start\_capture*. A tutorial on how to use the API Wizard and other TyphoonTest features is available on HIL Academy.



All signals are listed in this drop down menu, which is helpful to find various signal names for the capture.

The selected fault is then enabled halfway through the capture and the capture results are taken after it is finished.

```
268     cap.wait(secs=time_before_fault)
269
270     hil.set_contactor(name=f'{fault}.enable',
271                      swControl=True,
272                      swState=True,
273                      )
274
275     df = cap.get_capture_results(wait_capture=True)
```

- The `get_capture_results` function is responsible for returning the capture results as a Pandas DataFrame. This is useful incase we want to do anything else with the signals, such as outputting them as a .csv file.

At this point, all the signals that were included in the capture function are available to look at over the 1 second period. Pytest uses the Allure framework to create html reports of tests after they are down. It is possible to look under `get_capture_results` of the html report to see all the captured signals and their respective graphs.

However, there is an additional function solely for the purpose of comparing multiple signals on a single plot. This is useful for viewing 3-phase signals as well as power signals.

There is also an additional function `to.csv()` dedicated to writing csv files using any selected signal. The `df.index` is a different format than usual so the command `df.index.total_seconds()` is meant to format the time into only seconds.

### 4.3 Setter Fixtures and Indirect Parameterization

You may have noticed that there are no setup and teardown lines for the other 3 fixtures that have not been discussed yet:

- `set_fault_type`
- `set_fault`
- `set_fault_resistance`

These fixtures also have the same name as the name of the parameters in the `test_faults` test case. The reason these fixtures exist are to solve a major problem within the test scripts.

By the time the `test_faults` test case begins, the `load_model` fixture setup has already finished. As explained earlier, the `load_model` fixture is responsible for setting the fault type and resistance. However, because the `load_model` fixture