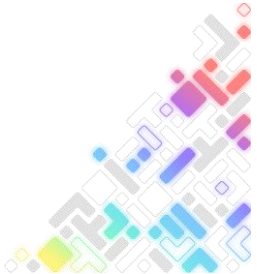
Zephyr® Project
Developer Summit

Pytest Tests in Twister

Maciej Perkowski (Nordic Semiconductor)

Overview

- Why?
- What’s pytest?
- How pytest is integrated with Twister?
- Examples
- Future?



Why:

- Improve quality of Zephyr
- On its own, twister supports:

  build → flash → parse output → give verdict

- But what if:
  - we want to use an existing python libraries in a test (e.g. REST API)
  - we need to communicate with DUT during the test (e.g. shell-based applications)?
  - a test requires some setup on the host side (e.g. connecting with a server)?
  - some additional tooling is to be used (e.g. a CLI-based tool, like MCUmgr or CAN interface)?
  - we want more than a single device (e.g. two devices talking to each other over Bluetooth)?

Pytest

"pytest is a mature full-featured Python testing tool that helps you write better programs.

The `pytest` framework makes it easy to write small, readable tests, and can scale to support complex functional testing for applications and libraries."

https://docs.pytest.org/en/8.0.x/

Main features

Plugins

- ✓ modules implementing hooks
- ✓ framework oriented (e.g. scope filtration, report generation, parallelization of test execution)
- ✓ **not that useful for us** (twister is the framework; pytest usability reduced to execute a detailed command constructed by Twister)

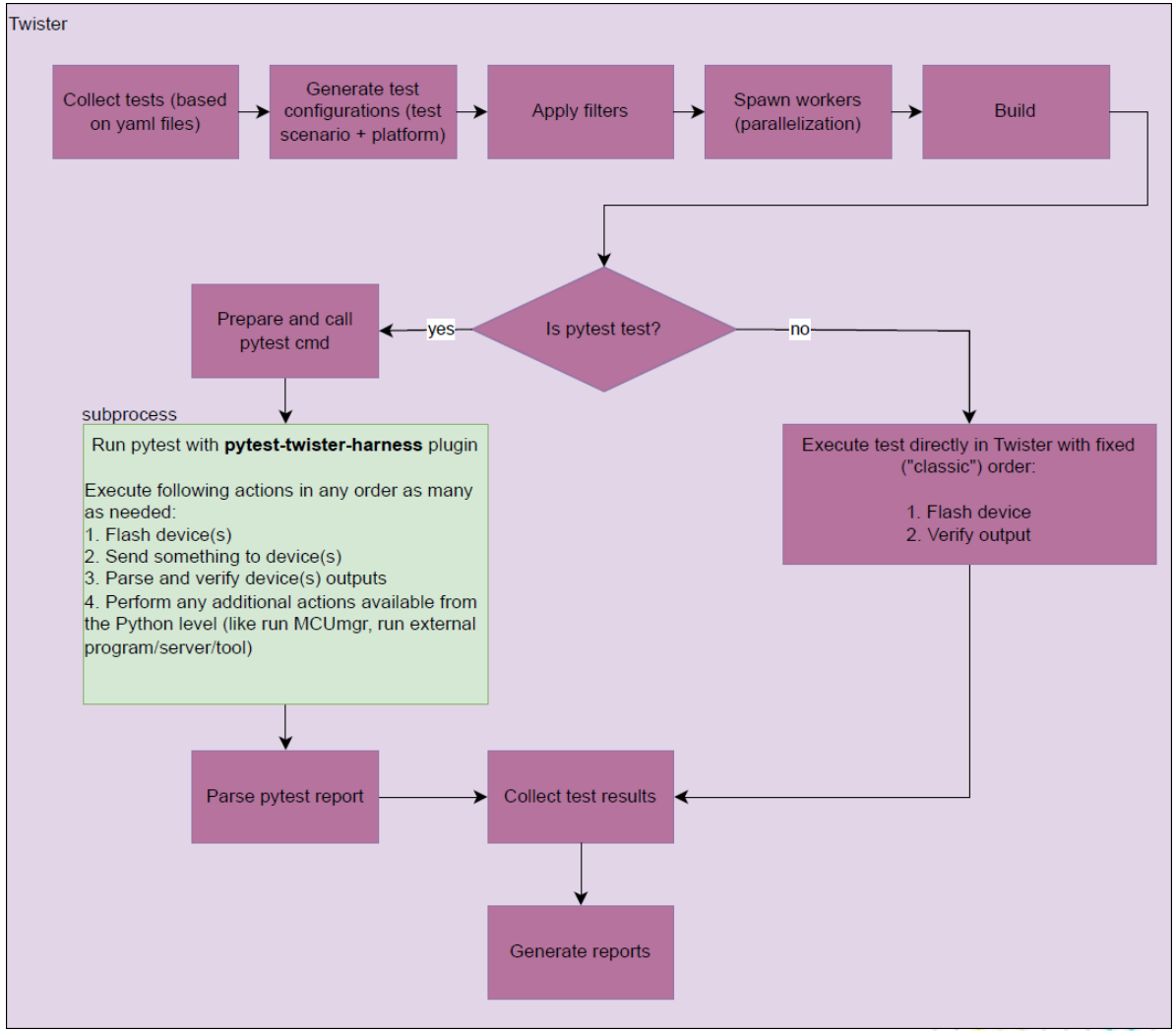
Fixtures

- ✓ initialize test functions
- ✓ test-oriented: a fixed baseline for reliable execution with consistent, repeatable, results
- ✓ explicit names, activated by declaring their use from test functions, modules, classes or whole projects
- ✓ modular: a fixture can use other fixtures
- ✓ scaling: in complexity and in scope
- ✓ easy and safely managed setup/teardown

Integration with Twister

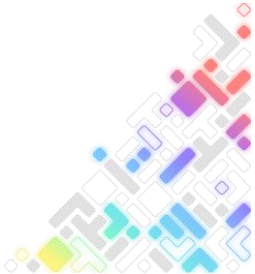
zephyr/tests/boot/with_mcumgr/testcase.yaml

```yaml
common:
  sysbuild: true
  platform_allow:
    - nrf52840dk/nrf52840
    - nrf5340dk/nrf5340/cpuapp
    - nrf9160dk/nrf9160
  integration_platforms:
    - nrf52840dk/nrf52840
  timeout: 600
  slow: true
tests:
  boot.with_mcumgr.test_upgrade:
    tags:
      - pytest
      - mcuboot
      - mcumgr
    harness: pytest
    harness_config:
      pytest_root:
        - "pytest/test_upgrade.py"
```



Seamless experience for users

```
14:52 $ ./scripts/twister -T samples/subsys/testsuite/pytest/shell/ -p native_posix –v
ZEPHYR_BASE unset, using "/home/maciej/zephyrproject/zephyr
"Renaming output directory to /home/maciej/zephyrproject/zephyr/twister-out.46
INFO    - Using Ninja..
INFO    - Zephyr version: v3.6.0-2048-g1ceceabad501
INFO    - Using 'zephyr' toolchain.
INFO    - Building initial testsuite list...
INFO    - Writing JSON report /home/maciej/zephyrproject/zephyr/twister-out/testplan.json
INFO    - JOBS: 4
INFO    - Adding tasks to the queue...
INFO    - Added initial list of jobs to queue

INFO    - 1/1 native_posix   samples/subsys/testsuite/pytest/shell/sample.pytest.shell PASSED (native 1.874s)

INFO    - 1 test scenarios (1 test instances) selected, 0 configurations skipped (0 by static filter, 0 at runtime).
INFO    - 1 of 1 test configurations passed (100.00%), 0 failed, 0 errored, 0 skipped with 0 warnings in 15.73 seconds
INFO    - In total 2 test cases were executed, 0 skipped on 1 out of total 707 platforms (0.14%)
INFO    - 1 test configurations executed on platforms, 0 test configurations were only built.
INFO    - Saving reports...
INFO    - Writing JSON report /home/maciej/zephyrproject/zephyr/twister-out/twister.json
INFO    - Writing xunit report /home/maciej/zephyrproject/zephyr/twister-out/twister.xml...
INFO    - Writing xunit report /home/maciej/zephyrproject/zephyr/twister-out/twister_report.xml...
INFO    - Run completed
```

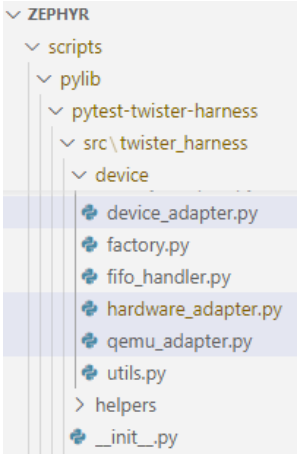



How to call just the pytest test?

- Call twister with extra verbosity (“–v –v”)
- Find line: “DEBUG - Running pytest command: “
- Copy – paste the command

```
DEBUG   - Running pytest command: export PYTHONPATH=/home/maciej/zephyrproject/zephyr/scripts/pylib/pytest-twister-harness/src:${PYTHONPATH} && pytest --twister-harness -s -v --build-dir=/home/maciej/zephyrproject/zephyr/twister-out/native_posix/samples/subsys/testsuite/pytest/shell/sample.pytest.shell --junit-xml=/home/maciej/zephyrproject/zephyr/twister-out/native_posix/samples/subsys/testsuite/pytest/shell/sample.pytest.shell/report.xml --log-file-level=DEBUG '--log-file-format=%(asctime)s.%(msecs)d:%(levelname)s:%(name)s: %(message)s' --log-file=/home/maciej/zephyrproject/zephyr/twister-out/native_posix/samples/subsys/testsuite/pytest/shell/sample.pytest.shell/twister_harness.log /home/maciej/zephyrproject/zephyr/samples/subsys/testsuite/pytest/shell/pytest --log-cli-level=DEBUG '--log-cli-format=%(levelname)s: %(message)s' --device-type=native -p twister_harness.plugin
```

Device Adapters



[*scripts\pylib\pytest-twister-harness\src\twister_harness\device\device_adapter.py*]

```python
class DeviceAdapter(abc.ABC):
    """
    This class defines a common interface for all device types (hardware,
    simulator, QEMU) used in tests to gathering device output and send data to
    it.

    def write(self, data: bytes) -> None:
        """Write data bytes to device."""
        if not self.is_device_connected():
            msg = 'No connection to the device'
            logger.error(msg)
            raise TwisterHarnessException(msg)
        self._write_to_device(data)

    @abc.abstractmethod
    def _write_to_device(self, data: bytes) -> None:
        """Write to device directly through serial, subprocess, FIFO, etc."""
```

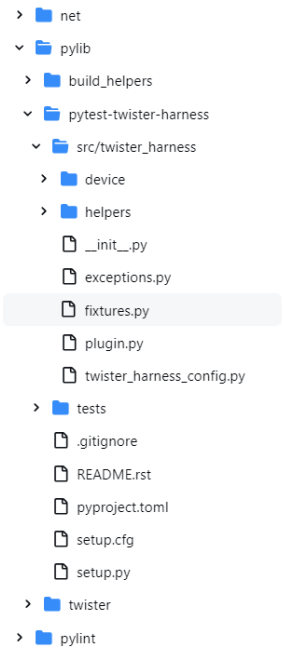
[*scripts\pylib\pytest-twister-harness\src\twister_harness\device\hardware_adapter.py*]

```python
    def _write_to_device(slf, data: bytes) -> None:
        self._serial_connection.write(data)
```

[*scripts\pylib\pytest-twister-harness\src\twister_harness\device\qemu_adapter.py*]

```python
    def _write_to_device(self, data: bytes) -> None:
        self._fifo_connection.write(data)
        self._fifo_connection.flush_write()
```

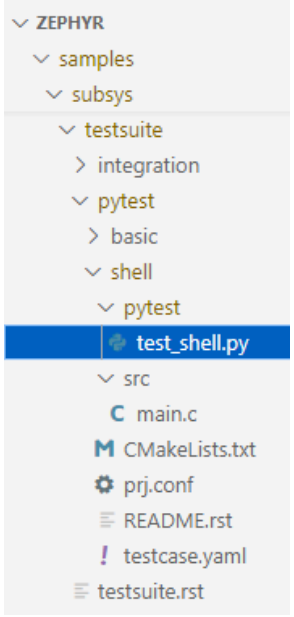
Fixtures




```python
@pytest.fixture(scope='session')
def device_object(twister_harness_config: TwisterHarnessConfig) -> Generator[DeviceAdapter, None, None]:
    """Return device object - without run application."""
    device_config: DeviceConfig = twister_harness_config.devices[0]
    device_type = device_config.type
    device_class: Type[DeviceAdapter] = DeviceFactory.get_device(device_type)
    device_object = device_class(device_config)
    try:
        yield device_object
    finally:  # to make sure we close all running processes execution
        device_object.close()


def determine_scope(fixture_name, config):
    if dut_scope := config.getoption("--dut-scope", None):
        return dut_scope
    return 'function'


@pytest.fixture(scope=determine_scope)
def dut(request: pytest.FixtureRequest, device_object: DeviceAdapter) -> Generator[DeviceAdapter, None, None]:
    """Return launched device - with run application."""
    device_object.initialize_log_files(request.node.name)
    try:
        device_object.launch()
        yield device_object
    finally:  # to make sure we close all running processes execution
        device_object.close()
```

scripts/pylib/pytest-twister-harness/src/twister_harness/fixtures.py

Example: shell interactions




```python
# Copyright (c) 2023 Nordic Semiconductor ASA
#
# SPDX-License-Identifier: Apache-2.0

import logging

from twister_harness import Shell

logger = logging.getLogger(__name__)


def test_shell_print_help(shell: Shell):
    logger.info('send "help" command')
    lines = shell.exec_command('help')
    assert 'Available commands:' in lines, 'expected response not found'
    logger.info('response is valid')


def test_shell_print_version(shell: Shell):
    logger.info('send "kernel version" command')
    lines = shell.exec_command('kernel version')
    assert any(['Zephyr version' in line for line in lines]), 'expected response not found'
    logger.info('response is valid')
```


samples\subsys\testsuite\pytest\shell\pytest\test_shell.py

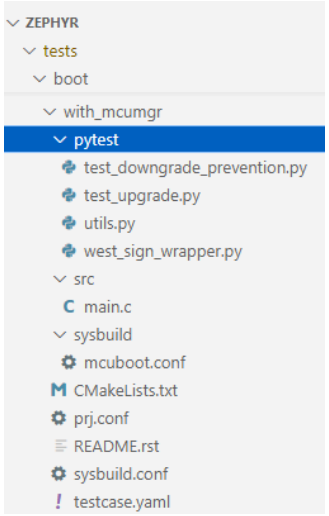
Example: shell interactions

```
DEBUG   - PYTEST:
samples/subsys/testsuite/pytest/shell/pytest/test_shell.py::test_shell_print_help
DEBUG   - PYTEST: ------------------------------ live log setup
-------------------------------
DEBUG   - PYTEST: DEBUG: Get device type "hardware"
DEBUG   - PYTEST: DEBUG: Opening serial connection for /dev/ttyACM0
DEBUG   - PYTEST: DEBUG: Flashing command:
/home/maciej/.pyenv/versions/zephyr38/bin/west flash --skip-rebuild --build-dir
/home/maciej/zephyrproject/zephyr/twister-
out/nrf52840dk_nrf52840/samples/subsys/testsuite/pytest/shell/sample.pytest.shell
DEBUG   - PYTEST: DEBUG: Flashing finished
DEBUG   - PYTEST: INFO: Wait for prompt
DEBUG   - PYTEST: DEBUG: Got prompt
DEBUG   - PYTEST: ------------------------------ live log call
-------------------------------
DEBUG   - PYTEST: INFO: send "help" command
DEBUG   - PYTEST: DEBUG: #: uart:~$ help
DEBUG   - PYTEST: DEBUG: #: Please press the <Tab> button to see all available
commands.
DEBUG   - PYTEST: DEBUG: #: You can also use the <Tab> button to prompt or auto-
complete all commands or its subcommands.
DEBUG   - PYTEST: DEBUG: #: You can try to call commands with <-h> or <--help>
parameter for more information.
DEBUG   - PYTEST: DEBUG: #: Shell supports following meta-keys:
DEBUG   - PYTEST: DEBUG: #: Ctrl + (a key from: abcdefklnpuw)
DEBUG   - PYTEST: DEBUG: #: Alt  + (a key from: bf)
DEBUG   - PYTEST: DEBUG: #: Please refer to shell documentation for more details
```

```python
# Copyright (c) 2023 Nordic Semiconductor ASA
#
# SPDX-License-Identifier: Apache-2.0

import logging

from twister_harness import Shell

logger = logging.getLogger(__name__)


def test_shell_print_help(shell: Shell):
    logger.info('send "help" command')
    lines = shell.exec_command('help')
    assert 'Available commands:' in lines, 'expected response not found'
    logger.info('response is valid')


def test_shell_print_version(shell: Shell):
    logger.info('send "kernel version" command')
    lines = shell.exec_command('kernel version')
    assert any(['Zephyr version' in line for line in lines]), 'expected
response not found'
    logger.info('response is valid')
```


samples\subsys\testsuite\pytest\shell\pytest\test_shell.py

Example: testing MCUboot with MCUmgr



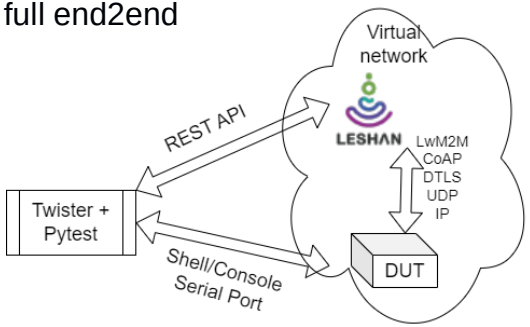

```
def test_downgrade_prevention(dut: DeviceAdapter, shell: Shell, mcumgr: MCUmgr):
        …
○       check_with_shell_command(shell, origin_version)
○       image_to_test = create_signed_image(dut.device_config.build_dir, '0.0.0+0’)
○       dut.disconnect()
○       mcumgr.image_upload(image_to_test)
○       mcumgr.reset_device()
        dut.connect()
○       output = dut.readlines_until('Launching primary slot application')
○       match_no_lines(output, ['Starting swap using move algorithm'])
○       match_lines(output, ['erased due to downgrade prevention'])
○       logger.info('Verify that the original APP is booted')
○       check_with_shell_command(shell, origin_version)
```

Example: end2end tests

LwM2M Interoperability tests using Leshan demo server
(tests/netlib/lwm2m/interopt)

Based on Open Mobile Alliance specification

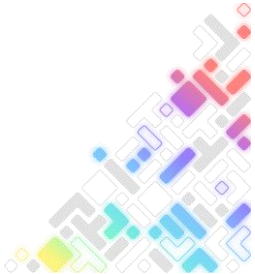
- Requires extra tools and environment configuration (docker image provided)
- full end2end




```
def test_LightweightM2M_1_1_int_102(shell: Shell, dut: DeviceAdapter, leshan:
Leshan, endpoint: str):
    """LightweightM2M-1.1-int-102 - Registration Update"""
    lines = shell.get_filtered_output(shell.exec_command('lwm2m read 1/0/1 -u32'))
    lifetime = int(lines[0])
    lifetime = lifetime + 10
    start_time = time.time() * 1000
    leshan.write(endpoint, '1/0/1', lifetime)
    dut.readlines_until(regex='.*net_lwm2m_rd_client: Update Done', timeout=5.0)
    latest = leshan.get(f'/clients/{endpoint}')
    assert latest["lastUpdate"] > start_time
    assert latest["lastUpdate"] <= time.time()*1000
    assert latest["lifetime"] == lifetime
    shell.exec_command('lwm2m write 1/0/1 -u32 86400')
```

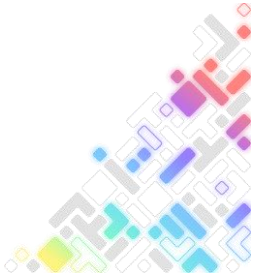
Future

- Adding more tests using already existing tools
  - Many low-hanging fruits for shell-using tests and samples (harness: keyboard)
  - Testing more protocols (e.g. CAN)
- Handling multiple images
  - Reusing in different tests
- Handling multiple devices
  - Devices talking to each other



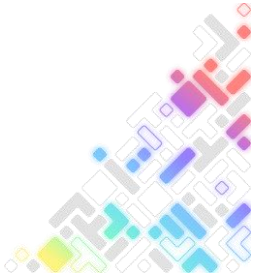
Summary

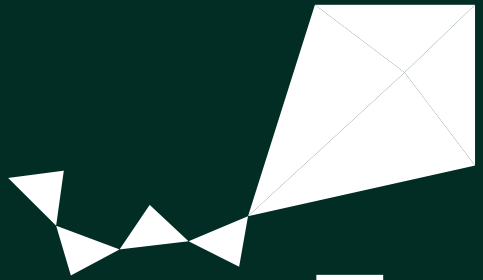
- Integration with pytest opened new levels of testing
- Low entry level
- Dut-type agnostic (thanks to Device Adapters)
- Easily extendable
- Fixtures:
  - Scalable
  - Configurable
  - Modular
  - Reusable



Helpful links

- https://docs.zephyrproject.org/latest/develop/test/pytest.html#
- https://github.com/zephyrproject-rtos/zephyr/tree/main/scripts/pylib/pytest-twister-harness
- https://github.com/zephyrproject-rtos/zephyr/issues/58288
- https://blog.golioth.io/automated-hardware-testing-using-pytest
- https://docs.pytest.org/en/6.2.x/fixture.html#what-fixtures-are





Zephyr® Project

Developer Summit