NVMe Compliance Suite

High Level Test Architecture

Various Ways to Write a Test

- Zero Dependency
  - Run just a test; you should expect it to pass
- Configuration Dependency
  - Must run a prior configuration which sets up resources for the target test
  - This “config” step is itself a test case
- Sequence Dependency
  - Must run all tests prior to the one your interested
  - Each test case adds new logic to create resources, or to place the hdw into a specific state for the following test(s) to rely upon.

A New Problem

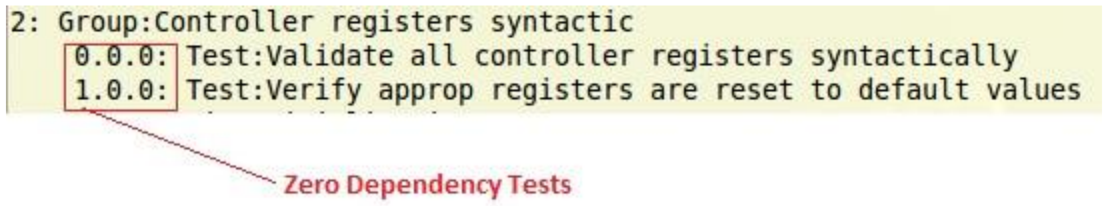
- The framework allowed developers to utilize all afore mentioned ways to write tests.
  - But how does the framework know how a developer decided to organize test dependencies?
  - In other words, if test x failed within group z, how would one run just that test, and what dependencies need to be run 1st?
    - Read the code to learn the dependencies
    - This is unacceptable

Resolution

- If a test fails, all we want to do is specify the failing test on the cmd line and let the framework figure out the dependencies and run them on our behalf.
- This requires borrowing from an existing framework construct
  - Redeploy the test numbering scheme
  - Test numbers now indicate test dependency.

Designate Zero Dependency

- Referenced by number x.0.0
  - Where x = {0…∞}
- Example:

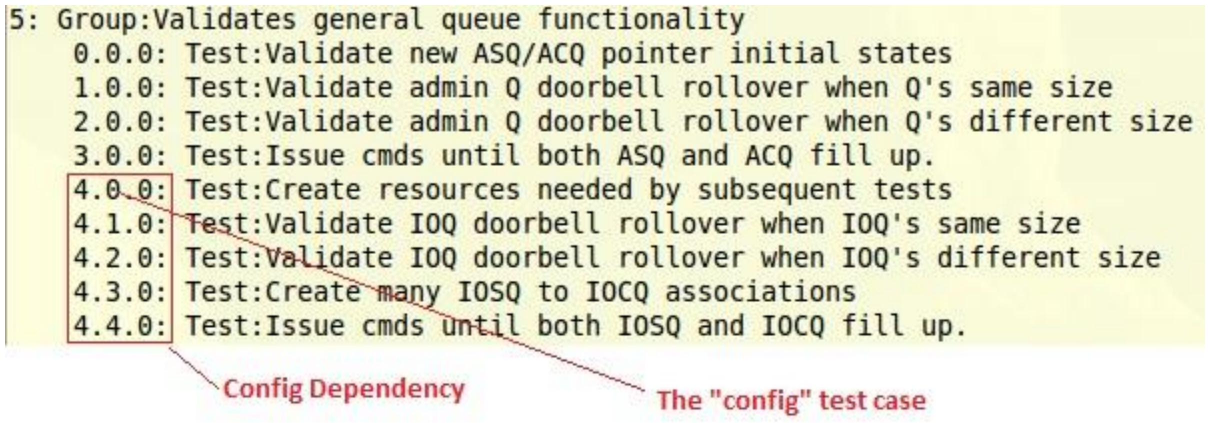


```
2: Group:Controller registers syntactic
    0.0.0: Test:Validate all controller registers syntactically
    1.0.0: Test:Verify approp registers are reset to default values
```

- Let's say test 1.0.0 fails
  - Execute: ./tnvme –test=2:1.0.0
  - Framework just runs a single test

Designate Config Dependency

- Referenced by number x.y.0
  - Where x = {0…∞}, y = {0…∞}
  - The “config” test must be designated by y=0
- Example:

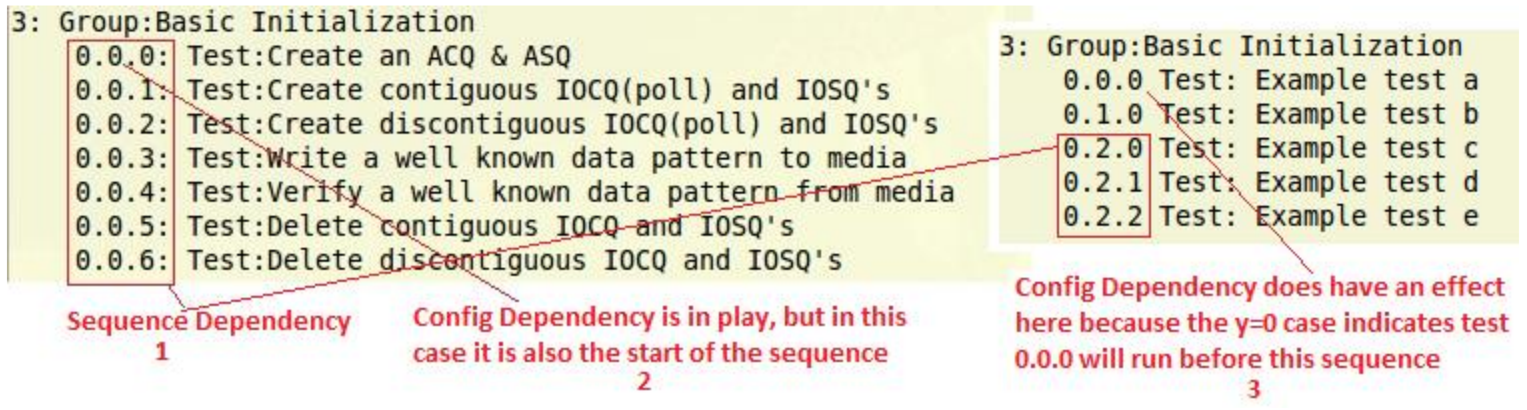


```
5: Group:Validates general queue functionality
    0.0.0: Test:Validate new ASQ/ACQ pointer initial states
    1.0.0: Test:Validate admin Q doorbell rollover when Q's same size
    2.0.0: Test:Validate admin Q doorbell rollover when Q's different size
    3.0.0: Test:Issue cmds until both ASQ and ACQ fill up.
    4.0.0: Test:Create resources needed by subsequent tests
    4.1.0: Test:Validate IOQ doorbell rollover when IOQ's same size
    4.2.0: Test:Validate IOQ doorbell rollover when IOQ's different size
    4.3.0: Test:Create many IOSQ to IOCQ associations
    4.4.0: Test:Issue cmds until both IOSQ and IOCQ fill up.
```

- Let’s say test 4.3.0 fails
  - Execute: ./tnvme –test=5:4.3.0
  - Framework 1st runs 4.0.0, and if successful runs 4.3.0

Designate Sequence Dependency

- Referenced by number x.y.z
  - Where x = {0…∞}, y = {0…∞}, z = {0…∞}
- Example:



```
3: Group:Basic Initialization
    0.0.0: Test:Create an ACQ & ASQ
    0.0.1: Test:Create contiguous IOCQ(poll) and IOSQ's
    0.0.2: Test:Create discontiguous IOCQ(poll) and IOSQ's
    0.0.3: Test:Write a well known data pattern to media
    0.0.4: Test:Verify a well known data pattern from media
    0.0.5: Test:Delete contiguous IOCQ and IOSQ's
    0.0.6: Test:Delete discontiguous IOCQ and IOSQ's
```

Sequence Dependency
1

Config Dependency is in play, but in this case it is also the start of the sequence
2

```
3: Group:Basic Initialization
    0.0.0 Test: Example test a
    0.1.0 Test: Example test b
    0.2.0 Test: Example test c
    0.2.1 Test: Example test d
    0.2.2 Test: Example test e
```

Config Dependency does have an effect here because the y=0 case indicates test 0.0.0 will run before this sequence
3

- Left example; Let's say test 0.0.3 fails
  - Execute: ./tnvme –test=3:0.0.3
  - Framework runs all 0.0.0, 0.0.1, 0.0.2, 0.0.3 in sequence
- Right example; Let's say test 0.2.1 fails
  - Execute: ./tnvme –test=3:0.2.1
  - Framework runs all 0.0.0, 0.2.0, 0.2.1 in sequence; note y=0 applies