


# CAV Structured Testing for Simulation

 The following pages have initially been designed/tested in ROS by Ansar Khan.  
They were then modified as appropriate to work with ROS2 + Gitlab CI by Sachin Fernando.


The [structured testing](#) framework is a method introduced for UWAFT to start ROS nodes. It serves as an extension of the existing [roslaunch framework](#).

The target use case for this tool is when a developer wants to test a part of the stack against a scenario that has been logged in bags. Certain nodes can be launched independently and their required topics can be made available from a given rosbag.


At a very high level this tool does 4 main tasks:

1. Start launch files
2. Play rosbags, while only publishing required topics from nodes launched in 1.
3. Watch the simulation and assess its performance

More specific details and design decisions are briefly described in the slides below.

 Slides and video use ROS1 implementation. Newest version has been modified for ROS2

A video demo is linked below. This video can also be found on the server: Z:\CAV\Technical Resources\Structured ROS Node Testing Demo

 This work is part of a ME 599 project, if you have any questions or are confused about something please feel free to reach out to Ansar Khan at anytime. I am always open to answering any questions or clearing things up.

[az3khan@uwaterloo.ca](mailto:az3khan@uwaterloo.ca)

[khanzansar@gmail.com](mailto:khanzansar@gmail.com)

## Prerequisites

- This is written assuming the reader is familiar with ROS2

## Modifying Launch Files

- Launch files follow the standard XML format as described here: <https://docs.ros.org/en/foxy/Tutorials/Intermediate/Launch/Creating-Launch-Files.html>
- Example Launch File

```
• <launch>

  <node pkg = "perception_kalman" exec ="perception_kalman" />

</launch>
```

## The .sim File

A sim file is the core of this system, this file defines the launch files to start, relevant topics, the corresponding bag files, and the observers to assess the quality of the sim. In the background this file is parsed as a standard YAML file.

**Launch files** are specified with their package and file name that is available in the roscore. This is the same way the file would be launched in a terminal (i.e `ros2 launch foo bar.launch`).

**Topics:** In order for a launch file to work with this structured testing framework it is expected that you explicitly define the topics they publish and the topics they subscribe to. This is to allow a rosbag to be built and analyzed before launch enabling the valid topics to be pulled from the bag if needed.

- `published_topics`: List of topics that are provided by the nodes defined in the launch file
- `required_topics`: List of topics subscribed to by the nodes in the launch file



NOTE: Currently this feature does not support parsing Published and Required Topics recursively from include tags. If a launch file is including other launch files its topics would have to be manually added to the parent launch file.

**Bag files** are specified as paths on the file system. Ensure that each topic required from the bags is present in exactly one bag file. The system will fail to launch if a required topic is either missing or present in multiple bags.

**(New) Test duration** specifies the time (in seconds) that the test runs for between shutting down and exiting the process. This added feature allows the test to run and the simresults file to be generated automatically (for CI purposes). That said, users may still shut down the test anytime with CTRL+C if running it manually.

**Observers** are specified using the format defined in the [CAV Simulation Observer Framework](#)

Example .sim file

```
launch_files:
- package: structured_testing
  file: testA.launch

- package: structured_testing
  file: testB.launch

- package: rospy_tutorials
  file: advanced_publish.launch

published_topics:
- /echo
- /foxtrot

required_topics:
- /alpha
- /bravo

bag_files:
- /home/parallels/Development/UWAFt/src/all_nodes.bag

test_duration:
- 10

observers:
- name: CmdVelX
  observerClass: InRangeObserver
  topic: "/cmd_vel"
  msgType: "geometry_msgs.msg.Twist"
  field: "linear.x"
  observerType: ALWAYS_TRUE
  minVal: 0
  maxVal: 10
```



1. Clone the [structured testing repo](#) and put it into the src folder of a catkin workspace
2. Build a `.sim` file with the required fields following the instructions above.
3. Run the `start_sim.py` script with a sim file as a command line argument (ex. `python3 start_sim.py ../test/example.sim`)
4. When you are ready to terminate the simulation kill the script with a sigterm (Ctrl-C), results of the simulation will be written to the same directly as the `.sim` file but with the `.simresults` extension



As of right now running structured tests through rosrn is not supported, invoke the script through python or directly run it. I.e (python3 start\_sim.py OR ./start\_sim.py



## (NEW) Integration with Gitlab CI

For this step to work the simulation must've been run with a `.simresults` file having been generated.

1. Run the `check_results.py` script with a `.simresults` file as a command line argument (ex. `python3 check_results.py ../test/example.simresults`)
2. Results check
  - a. If ALL tests pass: The result status is printed to the command line with exit code (0)
  - b. If a single test fails: The result status is printed to the command line with exit code (1)
3. The exit codes ensure that the Gitlab CI passes/fails as expected, which can be seen in the integrate-test phase of the `.gitlab-ci.yml` file



## Related articles

- [Lift Operation](#)
- [Active Variant Initialization Error](#)
- [Automation Scripting using Google APIs and Selenium WebDriver](#)
- [Google Form + Sheet + Mail + Script Integration](#)