Developer's Notes

Thespian
Python Actor System

By: Kevin Quick <quick@sparq.org>

2015 Sep 05 (#1.2)

Thespian Project

TheDoc-03

PUBLIC DOCUMENT

Contents

1 Source

1.1 Availability

Source code for Thespian is available at `https://github.com/kquick/Thespian` under the MIT license.

1.2 Documentation

Thespian documentation is maintained in the source tree under the `doc/` directory. Documentation is written in Emacs orgmode, which allows export as PDF, HTML, text, and other formats.

HTML output uses the `htmlize` package for syntax coloring, and the `bigblow` theme from Fabrice Niessen at `https://github.com/fniessen/org-html-themes`.

Documentation should be generated using the `doc/gen_html.sh` or `doc/gen_pdf.sh` bash script and specifying the source file for the document to be generated; the results are generated in the same `doc/` directory.

```
$ bash doc/gen_html.sh doc/using.org
```

1.3 Issues

Any issues should be posted via the main Thespian github page.

1.4 Contributions

Contributions (both source and documentation) should be in the form of pull requests. Contributions will not be accepted until the contributor has agreed to the contributor's agreement. Thespian uses the cla-assistant.io which will automatically check to see if a contributor has click-signed the contributor's agreement; as a contributor you will automatically be given a link to sign the agreement when you make your contribution.

2 Testing

2.1 Approach

While there are some Thespian tests that use mocking to reduce the effects of external elements, the main focus of Thespian testing is **functional** testing to make sure that the ActorSystem is providing proper functionality for the Actors that will be running. Writing mock'ed unit tests for Thespian is still a welcome contribution, we have just focused our primary efforts on the functional tests.

This functional approach means that the tests will actually create threads or processes (depending on the systembase) and that all systembases are tested on the same set of tests.

2.2 Test Duration

Because of the approach above running the tests can take some time. Test can even can hang if there is bad code or if system resources become exhausted (e.g. no more sockets available), and there can be spurious effects from the real world (e.g. timing failures if the local system is excessively busy).

Every effort is made to have tests run quickly and reliably, but validating functionality is the most important goal.

As of Thespian 3.7.0, the full test suite duration is approximately 1500 seconds.

2.3 Testing System Bases

Because all systembases are tested, it is necessary to accomodate the differences between those system bases. For most Actors, there should be no difference between systembases and they should be unaware of which one is running, but it is not practical for all systembases to support all features; the administrator starting the ActorSystem is responsible for starting a system appropriate to the task and Actors at hand.

Some of the system bases do not support some features (as detailed here in the Using Thespian guide) and so there

are inputs provided to the tests to indicate which are viable and/or stable. More on this in the Running Tests section below.

2.4 Running Tests

Tests are written to fairly conventional Python unittest standards without relying on plugins. Tests are normally run via the pytest package (run via `$ py.test`), and the latter provides various testing controls.

For simple, direct runs of specific tests, direct nose invocation of those tests is usually sufficient:

```
$ py.test thespian/test/testSimpleActor.py::TestASimpleSystem
```

As discussed above, all system bases are tested (by default) so the `simpleSystemBase` is usually covered by the `TestASimpleSystem` testcase so that it is tested before any of the more complex system bases (due to alphabetically appearing first).

The viability and focus of individual testcases is specified using methods from the `thespian/test/__init__.py` file:

unstable_test Specifies that the test is unstable for the specified base and should be skipped. The features tested may usually work, but are not reliably supported by the underlying base (e.g. the UDP transport does not provide delivery guarantees, so hi-traffic tests are marked with `unstable_test("multiprocUDPBase")`

```
unstable_test(asys, basename[, ...])
```

actor_system_unsupported Specifies that the functionality being tested is not supported by this particular system base and the test should be skipped for the indicated base(s).

```
actor_system_unsupported(asys, basename [,...])
```

When running the tests from the command line, the normal pytest-based commands can be used, with optional attribute specifiers to identify tests, categories ("Func" or "Unit"), and system bases. For example, to run all unit tests:

```
$ py.test -k Unit
```

The above is saved in `scripts/run_unit_tests.sh` for convenient re-use. In addition, the `scripts/run_main_functional_tes` file contains the following specification which is used to run all the stable functional tests for the standard set of testbases:

```
$ py.test -k Func
```

2.5 Writing Tests

When writing tests for Thespian, there is an `asys` fixture for pytest that provides an initialized ActorSystem for each system base. Writing the test to accept `asys` as a parameter will cause pytest to run the test once for each system base type.