# PyTango Status Report

Anton Joubert (SARAO)

Tango 2020 summer status update meeting

Wednesday, 10 June 2020

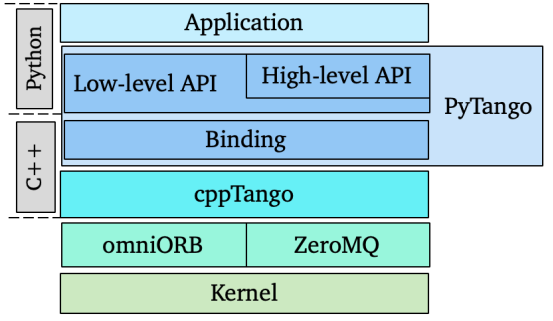
*

GitHub: ajoubertza/pytango-status-updates

Slides: https://ajoubertza.github.io/pytango-status-updates/

# PyTango? Quick reminder

- Python library
- Binding over the C++ tango library
- … using boost-python (future: pybind11)
- Relies on numpy
- Multi OS: Linux, Windows, Mac (with Docker…)
- Works on Python 2.7, 3.5, 3.6, 3.7, (probably 3.8)



# Recent releases

## Features in 9.3.1 and 9.3.2

- `MultiDeviceTestContext`
- `EnsureOmniThread`
- Windows wheels on PyPI
  - v9.3.1 working
  - v9.3.2 broken :-(
- 327 commits, 28 PRs, 29 issues

# Recent releases

## Fixes in 9.3.1 and 9.3.2

- Memory leak for DevEncoded attributes
- Dynamic enum attributes created without labels
- Python 3 issues
- Documentation
- Improvements for Linux packaging

Thanks to many first-time contributors: rhomspuron, asoderq, reszelaz and wyrdmeister!

# MultiDeviceTestContext

Like `DeviceTestContext`, but can launch multiple devices. In fact `DeviceTestContext` inherits from `MultiDeviceTestContext`.

Contributed by reszelaz - thanks!

Trivial example:

```
devices_info = (
    {"class": Device1,
     "devices": [{"name": "test/device1/1",
                  "properties": {"MyProperty": ["a", "b"]}}]
    },
    {"class": Device2,
     "devices": [{"name": "test/device2/1"}]
    }
)

def test_devices():
    with MultiDeviceTestContext(devices_info, process=True) as context:
        proxy1 = context.get_device("test/device1/1")
        proxy2 = context.get_device("test/device2/1")
        assert proxy1.attr1 == 1
        assert proxy2.attr2 == 2
```

# MultiDeviceTestContext

Detailed example available in pytango/examples

Contributed by DrewDevereux - thanks!

Using pytest, with `tango_context` a fixture that launches `MultiDeviceTestContext`

```
...
class TestMasterWorkerIntegration:
    def test_master_turn_worker_on(self, tango_context):
        master = tango_context.get_device("device/master/1")
        worker_1 = tango_context.get_device("device/worker/1")
        worker_2 = tango_context.get_device("device/worker/2")

        # check initial state: both workers are off
        assert worker_1.is_on == False
        assert worker_2.is_on == False

        # tell master to enable worker_1
        master.turn_worker_on(1)

        # check worker_1 is now on, and worker_2 is still off
        assert worker_1.is_on == True
        assert worker_2.is_on == False
...
```

# MultiDeviceTestContext

## Warning

If starting device more than once in the same process (e.g., once per test case), expect a segmentation fault!

`...TestContext(..., process=False)` is the default.

Options:

- `...TestContext(..., process=True)`
- nosetest can use `nose_xunitmp` plugin: `--with-xunitmp`
- pytest can use `pytest-forked` plugin: `--forked`

# EnsureOmniThread

Some issue reported when subscribing/unsubscribing to events from standard Python threads. Event channel not responding - see issue #307.

- cppTango uses omniorb threads (omnithreads), and their IDs are used for some thread locks.
- Main thread in PyTango device was always marked as an omnithread.
- Other user threads, typically `threading.Thread`, are not.

New `EnsureOmniThread` context handler added to help with this.

**Note:** Some of the issues this handler prevents have been fixed in cppTango, so you may be fine without it! Sorry, but it is confusing…

# EnsureOmniThread

## Example

```
import tango
from threading import Thread
from time import sleep

def thread_task():
    with tango.EnsureOmniThread():
        eid = dp.subscribe_event(
            "double_scalar", tango.EventType.PERIODIC_EVENT, cb)
        while running:
            print(f"num events stored {len(cb.get_events())}")
            sleep(1)
        dp.unsubscribe_event(eid)

cb = tango.utils.EventCallback()  # print events to stdout
dp = tango.DeviceProxy("sys/tg_test/1")
dp.poll_attribute("double_scalar", 1000)

thread = Thread(target=thread_task)
running = True
thread.start()
sleep(5)
running = False
thread.join()
```

# EnsureOmniThread

## Thread pools

Not sure how it could be used with `concurrent.futures.ThreadPoolExecutor` - see discussion here.

The gevent green mode also uses a thread pool, so similar problem...

# Asynchronous PyTango

**Also called green modes, checkout the docs:**

pytango.readthedocs.io/en/stable/green_modes/green.html

```
tango.GreenMode.Synchronous  # default
tango.GreenMode.Futures
tango.GreenMode.Gevent
tango.GreenMode.Asyncio
```

**`Asyncio` recommended for new projects** that want async features.

No plans to remove `Futures` or `Gevent`, but if support becomes problematic, `Asyncio` will get highest priority. This is because `asyncio` is the standard for Python.

Discussion notes

# Compatibility

## Python

- Maintain support for 2.7, 3.5, 3.6, and 3.7.
- Adding CI testing to verify 3.8.

Expect 3.8 to be fine, but don't have Python 3.8 Conda packages for CI dependencies yet.

# Compatibility

## cppTango

Up to now, matching *major.minor* releases of cppTango and PyTango should work.

Examples:

```
 cppTango | PyTango | Works?
----------|---------|-------
    9.3.4 |   9.3.2 | yes
    9.3.4 |   9.4.0 | maybe??
    9.4.1 |   9.4.0 | yes
```

**Note** cppTango 9.4.x will not be Application Binary Interface (ABI) compatible with cppTango 9.3.x, so not sure about PyTango.

After PyTango 9.4.0 is released, don't plan 9.3.x patches. Hopefully we can get PyTango to support both cppTango 9.3.x and 9.4.x.

# Packaging

## Installable versions

```
pip install pytango != apt-get install python-tango
```

PyPI has the latest

- but binding extension not compiled for Linux.
- binding is compiled and statically linked for Windows.

Conda

- v9.3.1 available on https://anaconda.org/tango-controls/pytango
- Busy with v9.3.2, moving to Github Actions here.

Linux packages

- The binding is already compiled code, so quick to install.
- Typically a few versions behind. Latest is v9.2.5?

**Volunteers?**: Pipelines to build Linux packages: Debian, Ubuntu, CentOS

# Packaging

## Docker images: SKA, with help from Tango Community

Dockerfiles: https://gitlab.com/ska-telescope/ska-docker

Images: https://nexus.engageska-portugal.pt

Based on Debian 10, latest use TangoSourceDistribution 9.3.4-rc4

```
$ docker pull nexus.engageska-portugal.pt/ska-docker/tango-pytango:9.3.2
```

**Warning:** This was pushed to docker hub 6 months ago as https://hub.docker.com/r/tangocs/tango-pytango, but not currently being updated from SKA build pipeline.

# New features?

## DeviceProxy support for TANGO_HOST with #dbase=no

```
class MyDevice(Device):
    @command(dtype_in=str, dtype_out=int)
    def ping_command(self, friend):
        dp = DeviceProxy(friend)
        return dp.ping()

def test_access():
    devices_info = ({"class": MyDevice,
                     "devices": [{"name": "my/dev/1"}, {"name": "my/dev/2"}]},)
    with MultiDeviceTestContext(devices_info) as context:
        dp1 = context.get_device("my/dev/1")
        dev2_fqdn = context.get_device_access("my/dev/2")
        assert dp1.ping_command(dev2_fqdn) < 1000  # works
        assert dp1.ping_command("my/dev/2") < 1000  # fails
```

`dev2_fqdn` something like `tango://172.17.0.3:48493/my/dev/2#dbase=no`

- modify test context to set `TANGO_HOST=172.17.0.3:48493#dbase=no` temporarily
- modify `DeviceProxy` to rewrite simple Tango names if `TANGO_HOST` has `#dbase=no`

# New features?

## More testing utilities

Goal is to make Tango devices more testable.

- pytest fixture like `MultiDeviceTestContext` example?
- Mock `DeviceProxy` class, including events?
- Support forwarded attributes with `DeviceTestContext`?
- No longer pursuing approach suggested as ICALECPS 2019: `@mock.patch('tango.server.Device', faketango.Device)`.

# New features?

## Pull docstrings into Command descriptions?

```
class PowerSupply(Device):

    @command(dtype_in=float, doc_in="Power supply output voltage")
    def voltage(self, set_point):
        self.set_hardware(set_point)
```

```
class PowerSupply(Device):

    @command(dtype_in=float)
    def voltage(self, set_point):
        """Power supply output voltage."""
        self.set_hardware(set_point)
```

Beneficial for IDEs, Sphinx autodoc, and more Pythonic. Already works this way for attributes.

Easiest to set `doc_in` and `doc_out` to `func.__doc__`.

# New features?

## Python logging as standard, sends to TANGO Logging Service?

Optionally add `init_logging` method and `logger` object on Device?

```
class PowerSupply(Device):

    @command
    def calibrate(self):
        self.logger.info('Calibrating...')
        # instead of info_stream('Calibrating...')
```

User could add/remove handlers, e.g., syslog, Elastic, or Tango Logging Service.

# PyTango development

## Hosting

- Repo: github.com/tango-controls/pytango
- Docs: pytango.readthedocs.io
- Continuous Integration: TravisCI, using Conda, Py 2.7, 3.5, 3.6, 3.7
- Windows packages: AppVeyor (TODO: dedicated `tango-controls` user)

## Issues

- Specific issues: report on GitHub - the more detail the better
- Questions: use the TANGO Forum

## Contributing

- Typical branched Git workflow. Main branch is `develop`
- Fork the repo, make it better, make a PR. Thanks!
- More info in how-to-contribute.

# Done! Any questions?

GitHub: ajoubertza/pytango-status-updates

Slides: https://ajoubertza.github.io/pytango-status-updates/