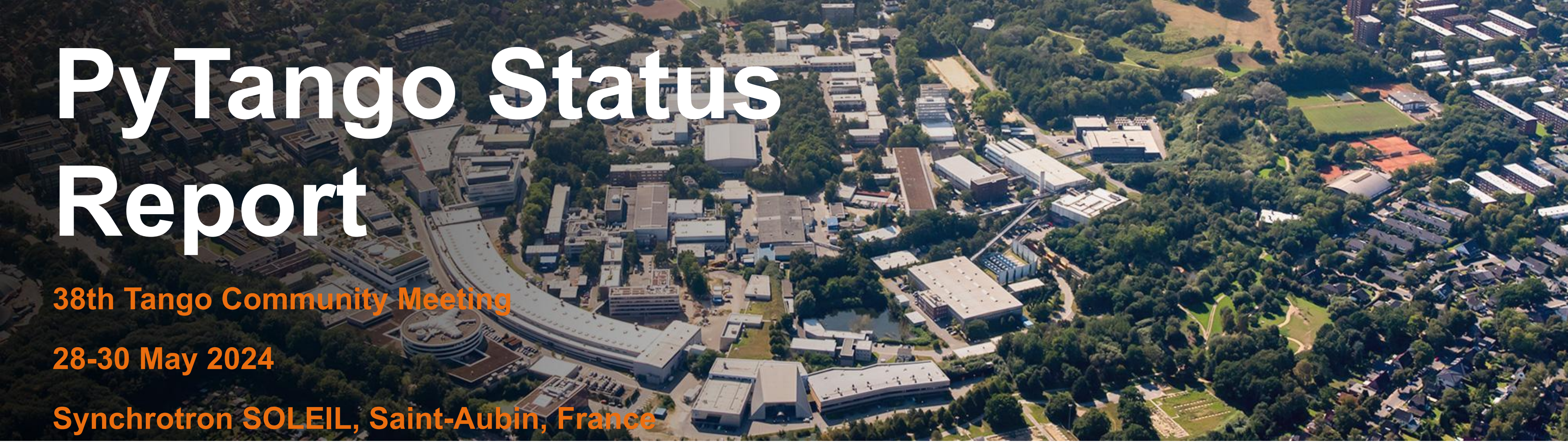


PyTango Status Report

38th Tango Community Meeting

28-30 May 2024

Synchrotron SOLEIL, Saint-Aubin, France

Yury Matveyev
Deutsches Elektronen-Synchrotron DESY

HELMHOLTZ

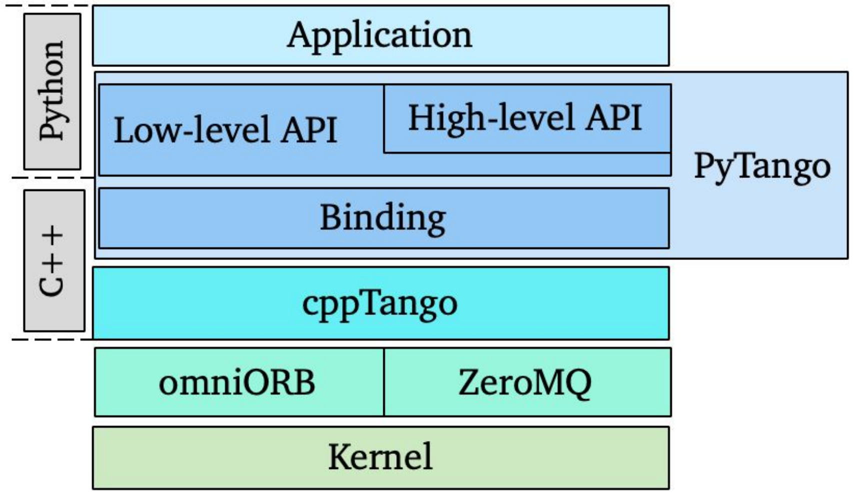
Acknowledgments

Co-maintainer, and contributor to these slides:

Anton Joubert

PyTango? Quick reminder

- ✔ Python library
- ✔ Binding over the C++ Tango library
- ✔ ... using boost-python
- ✔ Relies on numpy
- ✔ Multi OS: Linux, Windows and macOS
- ✔ Python 3.9 to 3.12 (next release 3.10 to 3.12)



PyTango Team

Regular attendees of our developers' meetings (twice a month - 1st and 3rd Thursdays, 15:00 CEST):

- Anton Joubert (MAX IV)
- Benjamin Bertrand (MAX IV)
- Corne Lukken (ASTRON)
- Jairo Moldes Fuentes (ALBA)
- Jose Antonio Ramos Andrades (ALBA)
- Mateusz Celary (S2Innovation)
- Mateusz Nabywaniec (S2Innovation)
- Thomas Ives (Observatory Sciences, SKAO)
- Thomas Juerges (SKAO)
- Ulrik Pedersen (Observatory Sciences, SKAO)
- Yury Matveev (DESY)

Join the #pytango channel on Tango Controls Slack.

Meeting minutes: https://gitlab.com/tango-controls/meeting-minutes/pytango

Current release - 9.5.1

March 2024

- **"Patch" release to pin Numpy dependency to 1.x**

- **Has a major regression in events pushing, better to use 9.5.0!**

- Asyncio green mode devices no longer crash when an attribute is read at the same time as an event is being pushed (MR, that causes regression)

- Improved some error message related to invalid types passed to DeviceProxy.
- Extended pydevd debugging and coverage to dynamic attributes and commands.
- High-level attribute reads using asyncio DeviceProxies can now be awaited.
- Numpy 1.20.0 no longer causes an import error.
- High-level Device class inheritance now supports class_property.

Current recommended release - 9.5.0

November 2023

- **Major release with (small) breaking API changes**

- The DevInt data type was removed because the corresponding DEV_INT type was removed from cppTango

- New features:
    - Short-name access can be used for (Multi)DeviceTestContext devices.
    - Support Tango server debugging with PyCharm, PyDev and VS Code (only static attributes)
    - use Python type hints to declare Device more easily (DevVarLongStringArray, DevVarDoubleStringArray missing)
    - Fixed various issues with DeviceProxy with non-synchronous green mode devices launched with TestContext.

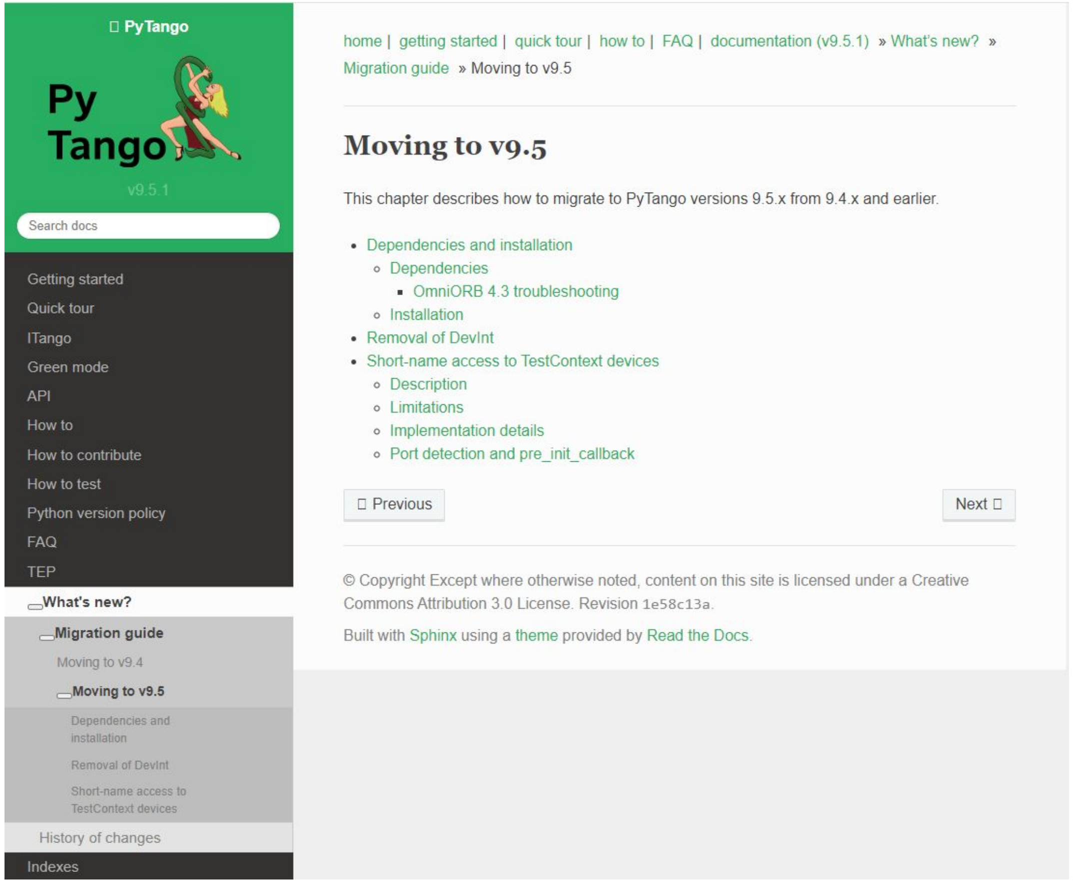
      This also fixes support for tests decorated with @pytest.mark.asyncio

Packages for 9.5.1 and 9.5.0

- Source on PyPI
- Binary wheels on PyPI
  - contain cppTango 9.5.0, omniorb, zmq, etc.
  - Windows: Python 3.9 to 3.12 (32-bit, 64-bit)
  - Linux: Python 3.9 to 3.12 (x86_64, i686, aarch64)
  - macOS: Python 3.9 to 3.12 (x86_64, arm64)
- Conda binary (pytango on conda-forge channel)
  - Python 3.9 to 3.12
  - Linux (x86_64, aarch64), Windows (64-bit), macOS (x86_64, arm64)
  - cppTango 9.5.0

Migration guide

See the new migration guide for the details of moving to 9.5.x



Moving to v9.5

This chapter describes how to migrate to PyTango versions 9.5.x from 9.4.x and earlier.

- Dependencies and installation
  - Dependencies
    - OmniORB 4.3 troubleshooting
  - Installation
- Removal of DevInt
- Short-name access to TestContext devices
  - Description
  - Limitations
  - Implementation details
  - Port detection and pre_init_callback

Previous release - 9.4.2

July 2023

➢ **Minor release, no new functionality**

- New python and NumPy version policy is implemented. Now only for Python > 3.9
- macOS wheels!
- DevEncoded attributes and commands read methods are now segfault (and memory leak) safe
- DevEncoded attributes and commands now decoded with utf-8 (as it was promised in documentation ☺️)
- DevEncoded attributes and commands can be extracted and written as str, bytes and bytesarray
- If string encoding with Latin-1 fails, UnicodeError will be raised instead of segfaulting

Highlights: short-name access to TestContext devices

```python
from tango import DeviceProxy
from tango.server import Device, attribute
from tango.test_context import MultiDeviceTestContext

class MyUselessDevice(Device):

    @attribute
    def attr(self) -> int:
        return 1

class MyOtherUselessDevice(Device):

    @attribute
    def attr(self) -> int:
        return DeviceProxy("test/device/1").attr # <---- not possible before


devices_info = ({"class": MyUselessDevice,      "devices": [{"name": "test/device/1"}]},
                {"class": MyOtherUselessDevice, "devices": [{"name": "test/device/2"}]},)

if __name__ == "__main__":
    with MultiDeviceTestContext(devices_info) as context:
        trl = "test/device/2" # <--- instead of context.get_device_access("test/device/2")
        print(DeviceProxy(trl).attr)
```

- ➢ Default behavior, can be disabled by setting *enable_test_context_tango_host_override* attribute to False before starting the TestContext
- ➢ Limitations:
  - ○ Group patterns (* wildcard) are not supported
  - ○ Launching two TestContexts in the same process will not work correctly without FQTRLs.

Highlights: Python type hints to declare Device

- easily declared and more readable Devices, allows doing static type checks (partially) with tools like mypy

```python
from numpy.random import random_sample

from tango import AttrWriteType
from tango.server import Device, attribute, command,
device_property


class SomeDevice(Device):

    host = device_property(dtype=str)

    noise = attribute(dtype=((float,),), max_dim_x=1024,
max_dim_y=1024)

    def read_noise(self):
        return random_sample((1024, 1024))

    @attribute (dtype=float,
access=AttrWriteType.READ_WRITE)
    def current(self):
        return self._my_current

    @current.setter
    def set_current(self, current):
        self._my_current = current

    @command(dtype_in=bool, dtype_out=bool)
    def output_on_off(self, on_off):
        self._output_on = on_off
        return self._output_on
```

⇨

```python
from numpy.random import random_sample

from tango import AttrWriteType
from tango.server import Device, attribute, command,
device_property


class SomeDevice(Device):

    host: str = device_property()

    noise = attribute(max_dim_x=1024, max_dim_y=1024)

    def read_noise(self) -> list[list[float]]:
        return random_sample((1024, 1024))

    @attribute (access=AttrWriteType.READ_WRITE)
    def current(self) -> float:
        return self._my_current

    @current.setter
    def set_current(self, current: float):
        self._my_current = current

    @command
    def output_on_off(self, on_off: bool) -> bool:
        self._output_on = on_off
        return self._output_on
```

Highlights: new build system using cmake

- ➢ extensions now built using PyPA's build module and scikit-build-core
- ➢ Windows builds now do not require black magic involved!
- ➢ build with debug symbols just by CL arguments (before setup.py had to be modified)
- ➢ running one test out-of-box (before setup.cfg always had to be modified)
- ➢ several cmake presets provided
- ➢ Much faster re-compilation, when developing extension code

Summury 9.4.1 -> 9.5.1

- 10 MRs in total - https://gitlab.com/tango-controls/pytango/-/releases/v9.5.1
- 48 MRs in total - https://gitlab.com/tango-controls/pytango/-/releases/v9.5.0
- 22 MRs in total - https://gitlab.com/tango-controls/pytango/-/releases/v9.4.2

Contributors since last year - thanks!

Anton Joubert, Jose A. Ramos, Benjamin Bertrand, Mateusz Nabywaniec, Mateusz Celary,

Ulrik Pedersen, Thomas Braun, Yury Matveyev.

Upcoming release: 10.0.0

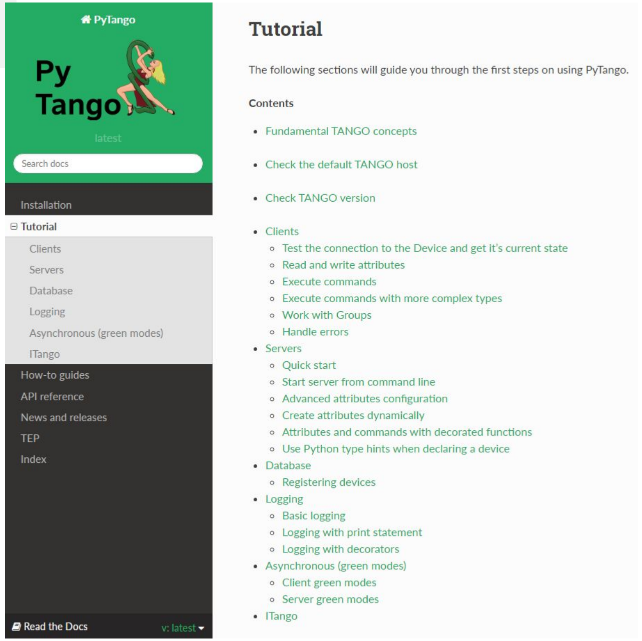
- ~ 1 month after cppTango 10.0.0
- Support for cppTango 10.0.0:
  - IDLv6
  - Distributed tracing via OpenTelemetry (separate talk by Anton Joubert)
  - Alarm event
  - DevInfo6
- Python 3.10 to 3.12 according to our version policy. But: we did not introduced any changes, which are incompatible with 3.9: should we still build the wheels for 3.9?
- Most probably no support of Numpy 2.0 due to Boost.Python. Is it dead? Should we urgently start migration to pybind11?
- Moved to C++ 17

Upcoming release: 10.0.0

- Revert events push regression (but introduce original bug back)
- Enable to push events with Python's exceptions:

```
@command
def send_change_event_with_exception(self):
    self.push_change_event("attr", Exception("test exception"))
```

- Fix segfault in push archive event with no data for attributes != state or status
- Redirect server errors to stderr (instead of stdout)
- Re-arrange docs
- New Asyncio servers implementation



Tutorial

The following sections will guide you through the first steps on using PyTango.

Contents

- Fundamental TANGO concepts
- Check the default TANGO host
- Check TANGO version
- Clients
  - Test the connection to the Device and get it's current state
  - Read and write attributes
  - Execute commands
  - Execute commands with more complex types
  - Work with Groups
  - Handle errors
- Servers
  - Quick start
  - Start server from command line
  - Advanced attributes configuration
  - Create attributes dynamically
  - Attributes and commands with decorated functions
  - Use Python type hints when declaring a device
- Database
  - Registering devices
- Logging
  - Basic logging
  - Logging with print statement
  - Logging with decorators
- Asynchronous (green modes)
  - Client green modes
  - Server green modes
- ITango

New Asyncio servers implementation

- ➢ Problem: original PyTango code was based of @asyncio.coroutine decorator, which was deprecated since Python 3.8 and removed in 3.11. In 3.12 generator-based coroutines were removed from constants, so they are not recognized as coroutines (e.g. coroutines.iscoroutine()) anymore -> our code did not work.

- ➢ Solution: we would not convert sync functions to coroutines on-the-fly in future. Asyncio servers have to be written with “async def” method definition.

- ➢ The old code still stays, so no urgent changes. Every first run of sync user function we throw DeprecationWarning.
- ➢ As soon as old code breaks again: it will be removed and old servers won’t be able to run.
- ➢ Bonuses:
  - ○ a lot of asyncio-related bugs were fixed: dynamic command green_mode, command is allowed green_mode, log decorators, pre_init_callback, post_init_callback preserve return coroutine if wrapping coroutine, etc.
  - ○ AsyncioExecutor now ensures OmniThread: dedicated PyTangoThreadPoolExecutor, can be used by user too

Upcoming release: 10.0.0

Hosting

- Repo: gitlab.com/tango-controls/pytango.
- Docs: pytango.readthedocs.io.
- Continuous Integration: GitLab CI (Micromamba Docker container),
- Linux + Windows: own runners, MacOs: GitLab runners

Release cycle

- At least twice per year.
- Aim for release within 1 month of cppTango releases.
- Release candidates are published - please help us test with your CI!
- Conda Forge packages are sometimes rebuilt to fix problems in dependencies.

PyTango development

Issues

- Questions: use the TANGO Forum.
- Specific issues: report on GitLab - the more detail the better (ideally, example code).

Contributing

- Please join in!
- Developers' meeting twice a month.
- Typical branched Git workflow. Main branch is develop
- Fork the repo, make it better, make an MR. Thanks!
- More info in how-to-contribute, and our webinar.

Thank you

Contact

Deutsches Elektronen-
Synchrotron DESY

www.desy.de

Yury Matveyev
Photon Science Experiment Control Group
yury.matveev@desy.de