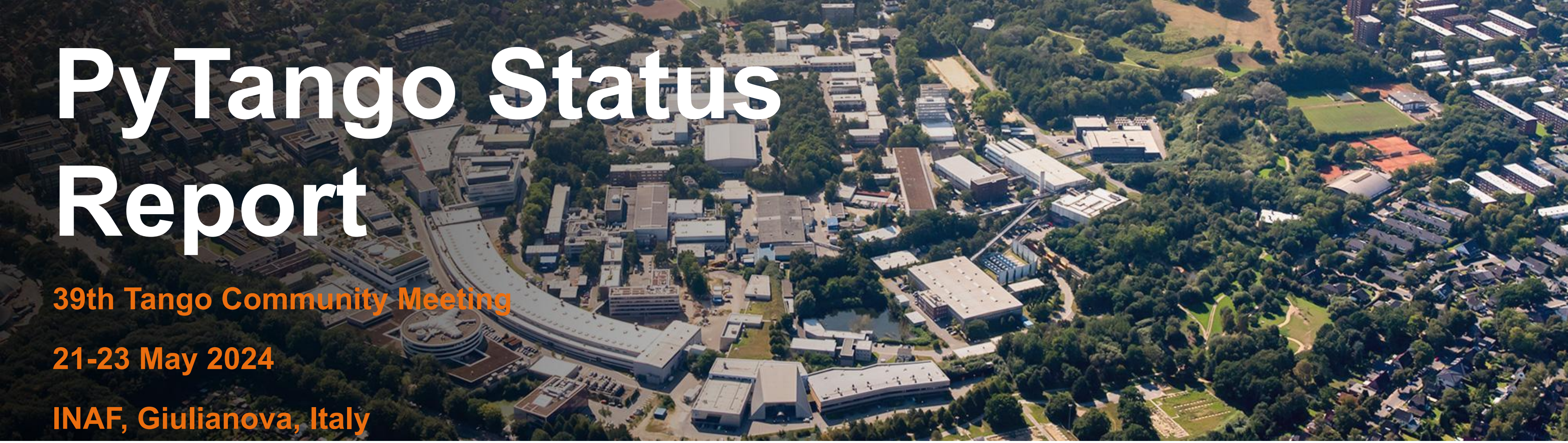


PyTango Status Report

39th Tango Community Meeting

21-23 May 2024

INAF, Giulianova, Italy

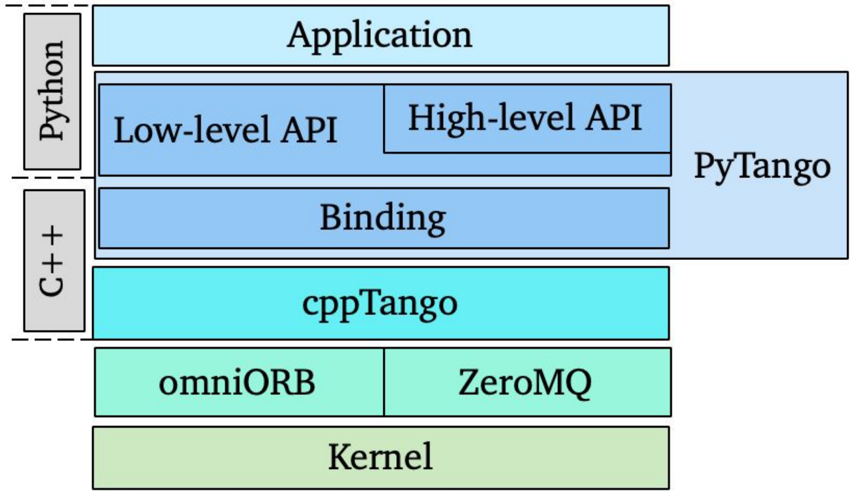
Yury Matveev
Deutsches Elektronen-Synchrotron DESY

Anton Joubert
MAX IV Laboratory

HELMHOLTZ

PyTango? Quick reminder

- ✔ Python library
- ✔ Binding over the C++ Tango library
- ✔ Multi OS: Linux, Windows and macOS
- ✔ Python 3.9 to 3.13



PyTango Team

Regular attendees of our developers' meetings, which started in September 2022:

- Anton Joubert (MAX IV)
- Benjamin Bertrand (MAX IV)
- Yury Matveev (DESY)
- Jose Antonio Ramos Andrades (ALBA)
- Jairo Moldes Fuentes (ALBA)
- Thomas Ives (Observatory Sciences, SKAO)
- Thomas Juerges (SKAO)
- Thomas Braun (Byte Physics)

We meet twice a month - 1st and 3rd Thursdays. 15:00 to 16:00 CET/CEST.

Join the #pytango channel on Tango Controls Mattermost.

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

Current release - 10.0.2

March 2025, minor release

- Wheels contain cppTango 10.0.2
- Fixed: occasional deadlock when a `Group` object that used events is destroyed
- Fixed: occasional segfault when an `AttrConfEventData` object are destroyed
- Fixed: segfault when pytest failure report tries to print device name
- `Group.command_inout()` and related methods accept simple data types, like `float` and `int`

Packages for 10.0.2

- Source on PyPI
- Binary wheels on PyPI
  - contain cppTango 10.0.2, omniorb, zmq, etc.
  - Windows: Python 3.9 to 3.13 (32-bit, 64-bit)
  - Linux: Python 3.9 to 3.13 (x86_64, i686, aarch64)
  - macOS: Python 3.9 to 3.13 (x86_64, arm64)
- Conda binary (pytango on conda-forge channel)
  - Python 3.9 to 3.13
  - Linux (x86_64, aarch64), Windows (64-bit), macOS (x86_64, arm64)
  - cppTango 10.0.x

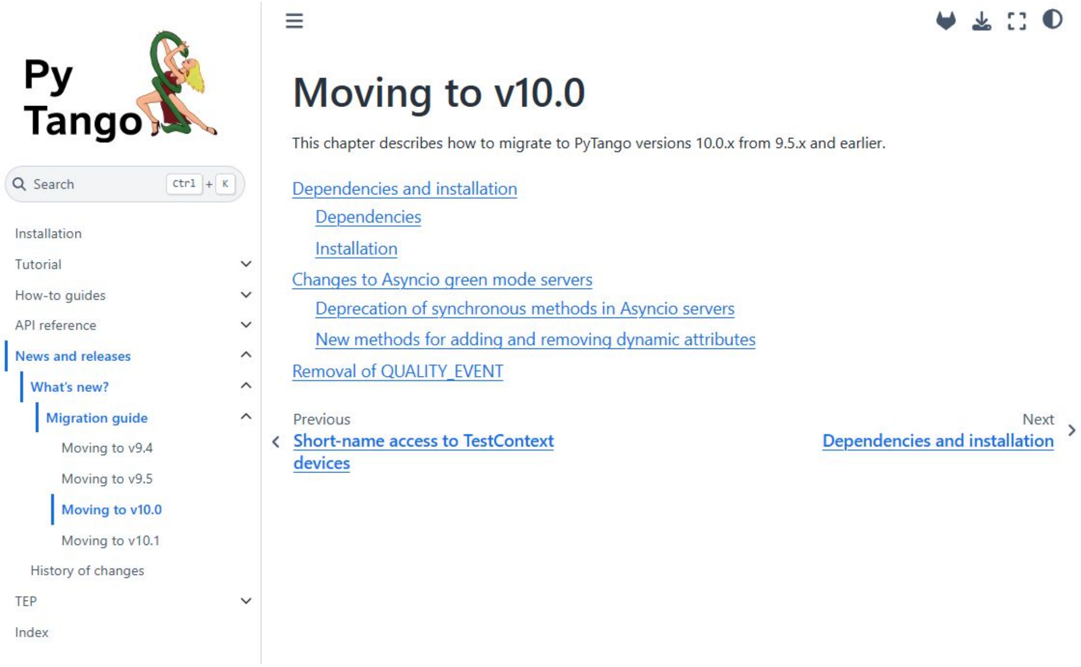
Previous release - 10.0.0

October 2024, major release

- Requires cppTango >= 10.0.0
- High-level Device and low-level LatestDeviceImpl classes now use Device_6Impl, with Tango IDLv6 interface
- NumPy 2.0 support
- OpenTelemetry support
- Alarm events
- Pydevd debugging (as well as coverage) now is extended to dynamic attributes and commands
- Stub file with typing information for improved autocompletion
- Segfault when Restart Command is used on a PyTango server fixed (Finally!)
- Asyncio mode considerably improved.
- Support for sync methods in asyncio mode is deprecated! As soon as Python will continue with clean-up of old asyncio code – it will be removed from PyTango!

Migration guide

See the new migration guide for the details of moving to 10.0.0 and 10.0.2



Moving to v10.0

This chapter describes how to migrate to PyTango versions 10.0.x from 9.5.x and earlier.

- Dependencies and installation
  - Dependencies
  - Installation
- Changes to Asyncio green mode servers
  - Deprecation of synchronous methods in Asyncio servers
  - New methods for adding and removing dynamic attributes
- Removal of QUALITY_EVENT

Previous: Short-name access to TestContext devices

Next: Dependencies and installation

Summary with last year:

- 45 MRs in total - https://gitlab.com/tango-controls/pytango/-/releases/v10.0.2
- 104 MRs in total - https://gitlab.com/tango-controls/pytango/-/releases/v10.0.0
- 16 MRs up to now for 10.1

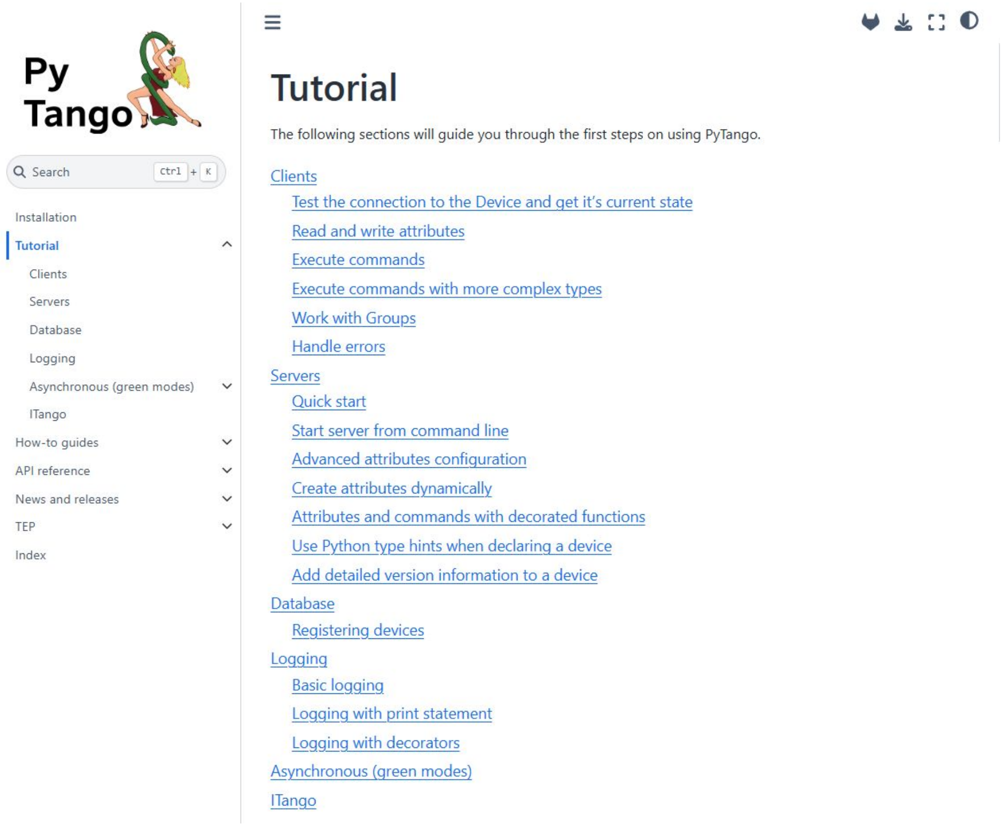
- Coverage of python code increased from 60 up to 74% (81% without Databaseds and deprecated code)
- Coverage of cpp code added (now 83%)

Contributors since last year:

Anton Joubert, Benjamin Bertrand, Thomas Braun, Yury Matveyev, Rodrigo Tobar, Johan Forsberg,

Thomas Juerges, Mateusz Celary, Samuel Debionne, Jose A. Ramos, PhilIJC

Highlights: better documentation



Py Tango

Search ctrl + K

Installation
Tutorial
Clients
Servers
Database
Logging
Asynchronous (green modes)
ITango
How-to guides
API reference
News and releases
TEP
Index

Tutorial

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

- 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
  - Add detailed version information to a device
- Database
  - Registering devices
- Logging
  - Basic logging
  - Logging with print statement
  - Logging with decorators
- Asynchronous (green modes)
- ITango

- A lot of useful information moved from endless unstructured “How-to” to Tutorials and categorized.

Enjoy!

Highlights

- Device description, default status and state can be set by class variables
- Events can be pushed with Python exceptions directly

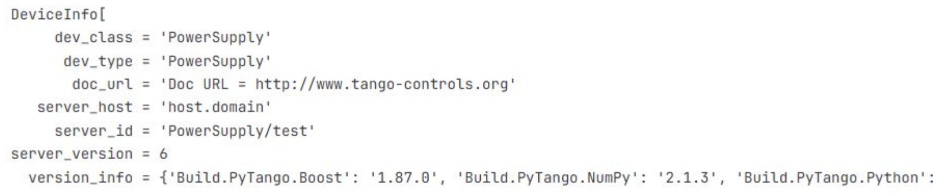
```python
from tango import Except, DevFailed, DevState
from tango.server import Device, attribute, command


class SomeDevice(Device):

    def init_device(self):
        super().init_device()
        self.set_change_event("attr", True, False)
        self.set_state(DevState.ON)
        self.set_status("Device is functional")


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

    @command
    def push_example(self):
        try:
            Except.throw_exception("Test_Reason",
                                   "a description",
                                   "PushChangeEventEx")
        except DevFailed as ex:
            self.push_change_event("attr", ex)
```

⇨

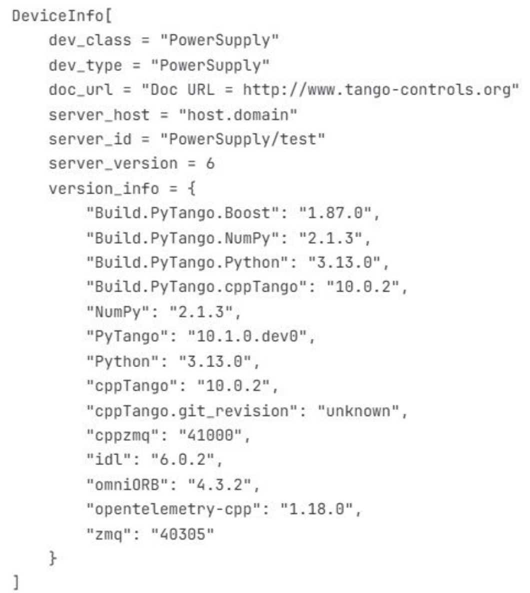
```python
from tango import DevState
from tango.server import Device, attribute, command


class SomeDevice(Device):

    DEVICE_CLASS_DESCRIPTION = "This is a test Tango device"
    DEVICE_CLASS_INITIAL_STATE = DevState.ON
    DEVICE_CLASS_INITIAL_STATUS = "Device is functional"

    def init_device(self):
        super().init_device()
        self.set_change_event("attr", True, False)

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

    @command
    def pust_example(self):
        self.push_change_event("attr",
                               RuntimeError("Test Reason"))
```

Upcoming release: 10.1.0

- ~ 1 month after cppTango 10.1.0
- Better pprint of PyTango structures



```
DeviceInfo[
     dev_class = 'PowerSupply'
      dev_type = 'PowerSupply'
       doc_url = 'Doc URL = http://www.tango-controls.org'
   server_host = 'host.domain'
     server_id = 'PowerSupply/test'
server_version = 6
  version_info = {'Build.PyTango.Boost': '1.87.0', 'Build.PyTango.NumPy': '2.1.3', 'Build.PyTango.Python':
```



```
DeviceInfo[
    dev_class = "PowerSupply"
    dev_type = "PowerSupply"
    doc_url = "Doc URL = http://www.tango-controls.org"
    server_host = "host.domain"
    server_id = "PowerSupply/test"
    server_version = 6
    version_info = {
        "Build.PyTango.Boost": "1.87.0",
        "Build.PyTango.NumPy": "2.1.3",
        "Build.PyTango.Python": "3.13.0",
        "Build.PyTango.cppTango": "10.0.2",
        "NumPy": "2.1.3",
        "PyTango": "10.1.0.dev0",
        "Python": "3.13.0",
        "cppTango": "10.0.2",
        "cppTango.git_revision": "unknown",
        "cppzmq": "41000",
        "idl": "6.0.2",
        "omniORB": "4.3.2",
        "opentelemetry-cpp": "1.18.0",
        "zmq": "40305"
    }
]
```

- User’s callback method will be dereferenced immediately after event unsubscription
- All known memory leaks fixed (and checked in CI)
- `get_client_ident` will be available in a `Device`
- GIL will be released in more `DeviceProxy` and `Device` methods

BIG news: Boost.Python -> PyBind11 accomplished !

Why?

- Pybind11 offers C++17 support (e.g. Boost bindings does not compile with cppTango 10.1 anymore)
- Pybind11 is “alive” project with regular updates and support of new Python versions
- Pybind11 is a header-only library, mush easier compilation in Windows
- Pybind11 is easier to debug: typical frame is < 10 encapsulated calls, wrt up to 200 in Boost

Additional outcome:

- A lot of broken code was fixed and covered with tests
- Bindings are now better structured and formatted (with clang-format)
- And we know them now!
- Compilation of binding is now warning-free
- 11k fewer lines of code! (mostly for pipes)

Pybind11: API changes

1. **Pipes** and all related method **were removed**
2. **Enums:**
   - Identity comparison, `device.State() is DevState.ON`, does not work any more.

     You must use equality: `device.State() == DevState.ON`
   - `__repr__` : if you did `repr(DevState.ON)` with boost you got `"tango._tango.DevState.ON"` , now `"<DevState.ON: 0>"`
   - Integer value is retrieved with `.value`, instead of `.real`. Or just use `int(my_enum)`.
   - Enums is not inherited from `int` anymore, so all int-related methods (`.real`, `.imag`, `.numerator`, etc.) are gone
3. **dim_x** and **dim_y** kwargs for `Attribute.set_value`, `Attribute.set_value_date_quality`, `Device.push_<>_event`

   are no longer supported
4. All **docstrings** for classes, methods and enums in pybind11 **aren't mutable**
5. Vectors `StdStringVector`, `StdLongVector`, `StdDoubleVector` are now implicitly convertible to Python lists, no need to convert
6. `StdGroupAttrReplyVector`, `StdGroupCmdReplyVector`, `StdGroupReplyVector` aren't exported any more.

   Instead, user receives `list[GroupAttrReply]`, `list[GroupCmdReply]`, `list[GroupReply]`, respectively
7. Attribute configuration (`AttributeConfig`, `AttributeAlarm`, etc.) structs interface frozen

See the migration guide for v10.1

PyTango: new logo

PyTango

PyTango development

Issues

- Questions: use the Mattermost or 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 Matveev
Photon Science Experiment Control Group
yury.matveev@desy.de