Connecting things together

4th Tango Kernel Webinar PyTango

Overview and how to contribute

Anton Joubert - NRF/SARAO
Geoff Mant - STFC

23 June 2021

https://www.tango-controls.org

Agenda

Introduction

Repository overview

Dependencies

How to: set up a dev environment, run the tests, add a new test

Architecture overview

Practical example: code navigation while reading an attribute

Useful tips

Contribution workflow

Questions

Introduction

Python library

Binding over the C++ tango library

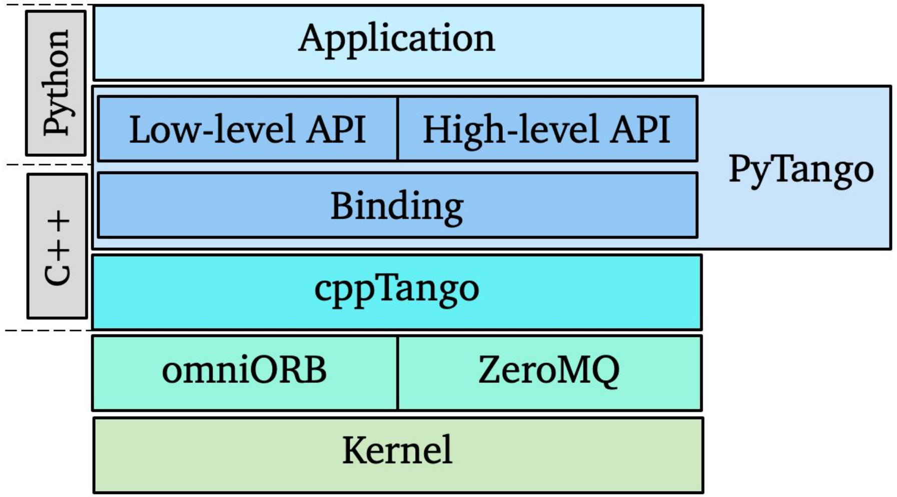
... using boost-python (future: pybind11)

Does not use omniorb Python library

Relies on numpy

Multi OS: Linux, Windows, MacOS (sort-of)

Works on Python 2.7, 3.5+



Introduction

< 2003 ?	Project started at SOLEIL
2005	Moved to ALBA. M. Taurel develops server.
2006	T. Coutinho main contributor.
2010	First package on pypi.org.
2012	High-level server API.
2013	Project moves with T. Coutinho to the ESRF.
2015	Included as Debian 8 package.
2016	PyTango 9 is released. V. Michel joins as maintainer.
2017	Welcome to Solaris.
2018	Welcome to SKA and institutes working on that project.
2019	A. Joubert joins as maintainer.
2021	Conda package on conda-forge (previously tango-controls).

* Welcome to all institutes, even if not mentioned!

Introduction

Original goal:

Provide a Python wrapper around the cppTango library

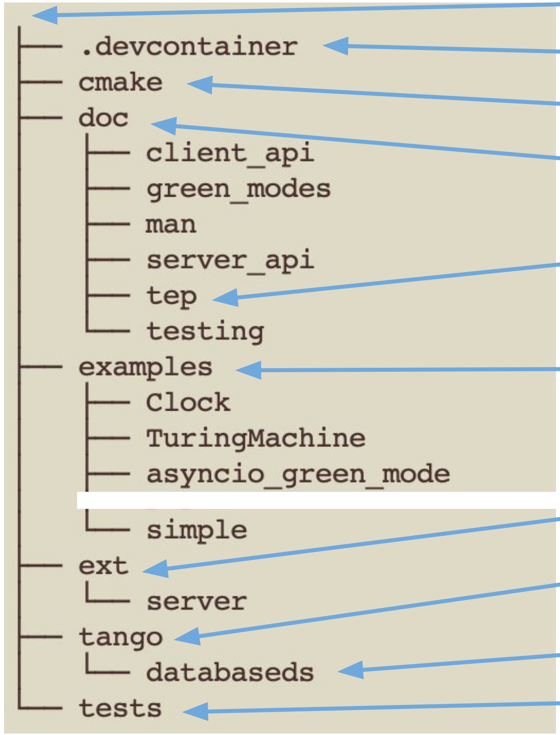
This resulted in the “low-level” Python API, closely matching cpp code.

Later:

Provide a Pythonic way of using Tango

This resulted in the “high-level” Python API, much nicer for Python programmers

Repository overview



```
├── .devcontainer
├── cmake
├── doc
│   ├── client_api
│   ├── green_modes
│   ├── man
│   ├── server_api
│   ├── tep
│   └── testing
├── examples
│   ├── Clock
│   ├── TuringMachine
│   ├── asyncio_green_mode

│   └── simple
├── ext
│   └── server
├── tango
│   └── databaseds
└── tests
```

Root has usual suspects: `setup.py`, CI yaml, license, etc.

Docker container for developers, editor configs, etc.

Cmake utilities (used for Windows compilation)

Sphinx docs

PyTango Extension Proposals (a bit like Python PEPs)

Various examples

C++ extension code

Python modules (used to be named `PyTango`)

Old Python-based DatabaseDS

Unit tests

Dependencies

OS dependencies:

- libtango >= 9.3, and its dependencies: omniORB4 and libzmq
- Boost.Python >= 1.33

Python dependencies:

- numpy >= 1.1
- six >= 1.10

Build dependencies:

- Setuptools
- Sphinx

Optional dependencies:

- futures
- gevent

How to set up a dev environment?

Clone the repo (or your fork)

```
git clone git@gitlab.com:tango-controls/pytango.git
```

Build a dev docker image in the `.devcontainer` folder (readme)

```
cd .devcontainer
export PYTHON_VERSION=3.8 TANGO_VERSION=9.3.4
docker build . -t pytango-dev:py${PYTHON_VERSION}-tango${TANGO_VERSION} \
               --build-arg PYTHON_VERSION --build-arg TANGO_VERSION
```

Run docker container, bind mount your source as a volume

```
docker run -it --rm --name pytango-dev -v ~/tango-src/pytango:/opt/pytango \
           pytango-dev:py3.8-tango9.3.4 /bin/bash
```

Inside the container build the extension, optionally run tests

```
cd /opt/pytango
python setup.py build
python setup.py test
```

How to set up a dev environment?

If you want to run PyTango scripts, pytest, or use from a Python session:

```
cd /opt/pytango
pip install -e .
```

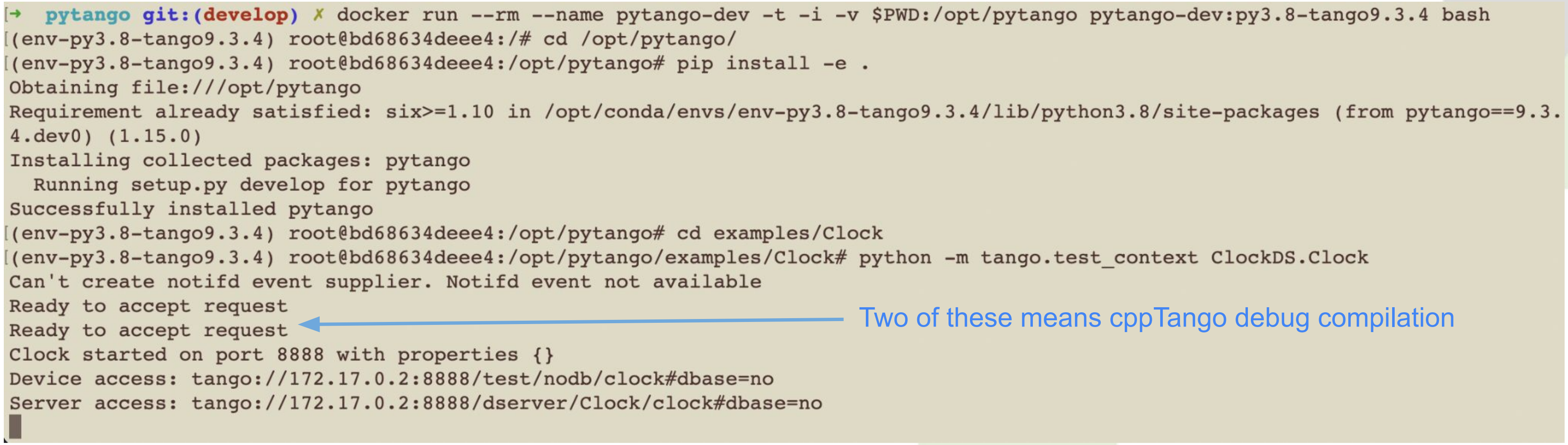
Configure your IDE:

PyCharm (professional)
VS Code (remote containers extension)

Details in the readme

How to set up a dev environment?

Running the example Clock device server in the container

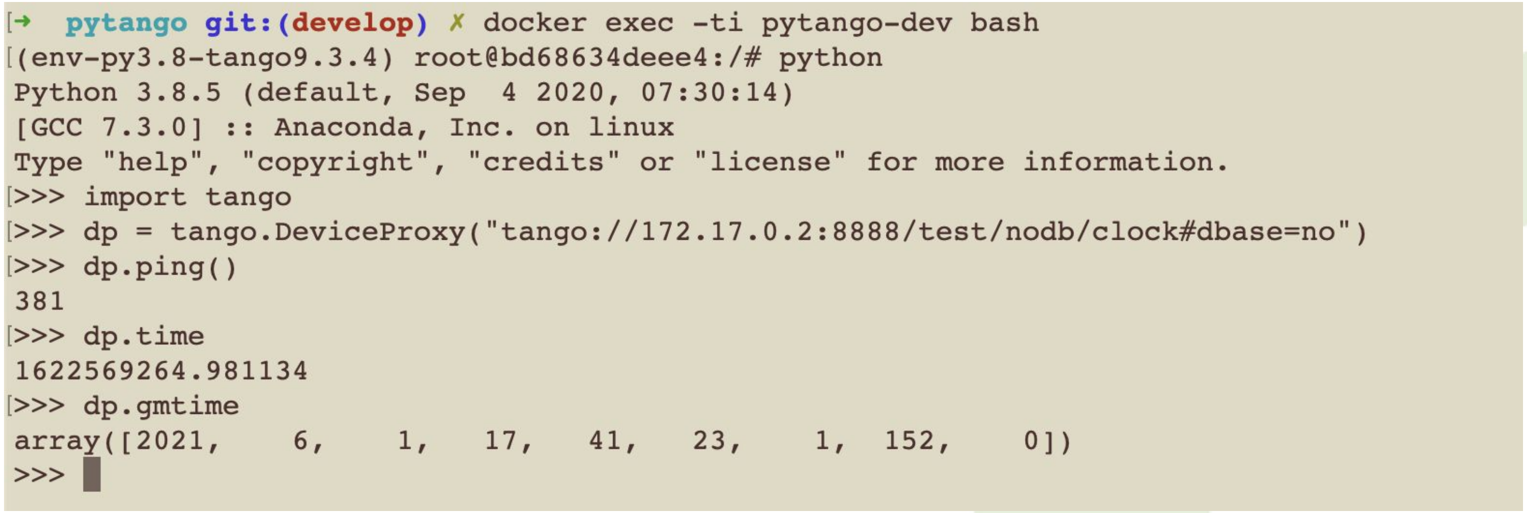


```
➜  pytango git:(develop) ✗ docker run --rm --name pytango-dev -t -i -v $PWD:/opt/pytango pytango-dev:py3.8-tango9.3.4 bash
(env-py3.8-tango9.3.4) root@bd68634deee4:/# cd /opt/pytango/
(env-py3.8-tango9.3.4) root@bd68634deee4:/opt/pytango# pip install -e .
Obtaining file:///opt/pytango
Requirement already satisfied: six>=1.10 in /opt/conda/envs/env-py3.8-tango9.3.4/lib/python3.8/site-packages (from pytango==9.3.
4.dev0) (1.15.0)
Installing collected packages: pytango
  Running setup.py develop for pytango
Successfully installed pytango
(env-py3.8-tango9.3.4) root@bd68634deee4:/opt/pytango# cd examples/Clock
(env-py3.8-tango9.3.4) root@bd68634deee4:/opt/pytango/examples/Clock# python -m tango.test_context ClockDS.Clock
Can't create notifd event supplier. Notifd event not available
Ready to accept request
Ready to accept request
Clock started on port 8888 with properties {}
Device access: tango://172.17.0.2:8888/test/nodb/clock#dbase=no
Server access: tango://172.17.0.2:8888/dserver/Clock/clock#dbase=no
```

Two of these means cppTango debug compilation

How to set up a dev environment?

Connect to the example Clock device from another container shell



```
[➜  pytango git:(develop) ✗ docker exec -ti pytango-dev bash
[(env-py3.8-tango9.3.4) root@bd68634deee4:/# python
Python 3.8.5 (default, Sep  4 2020, 07:30:14)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
[>>> import tango
[>>> dp = tango.DeviceProxy("tango://172.17.0.2:8888/test/nodb/clock#dbase=no")
[>>> dp.ping()
381
[>>> dp.time
1622569264.981134
[>>> dp.gmtime
array([2021,    6,    1,   17,   41,   23,    1,  152,    0])
>>> 
```

How to run the tests?

Run full test suite (pip install required)

```
pytest
```

Run a single test

```
pytest -k test_async_command_polled[int]
```

Run a test and enter PDB on the first failure

```
pytest -k test_async_command_polled[int] --pdb
```

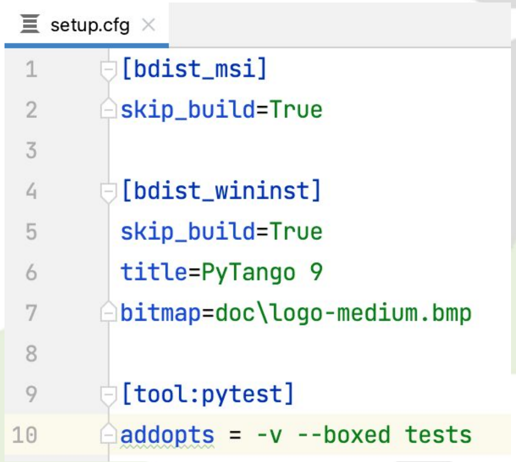
How to run the tests?

If running individual tests from PyCharm we need to edit `setup.cfg`:

  remove `tests` from pytest options, so not all tests are run.

The `--boxed` option runs each test in a new process as the `DeviceTestContext` can only be used once

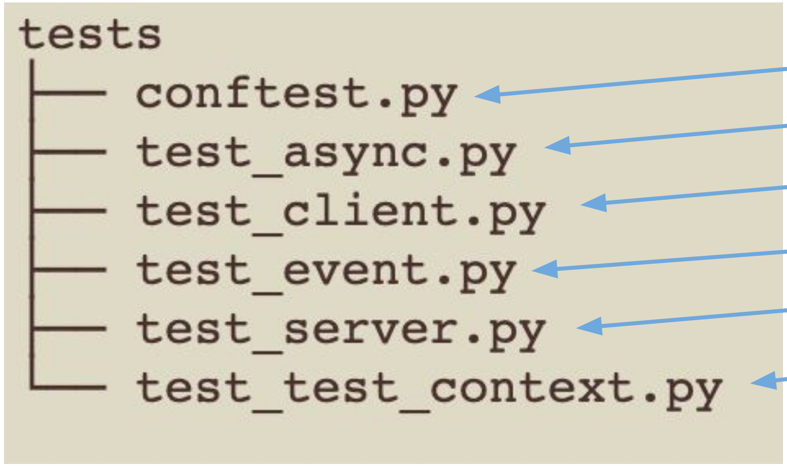
Note: the `--boxed` option is not supported on Windows


setup.cfg

```
[bdist_msi]
skip_build=True

[bdist_wininst]
skip_build=True
title=PyTango 9
bitmap=doc\logo-medium.bmp

[tool:pytest]
addopts = -v --boxed tests
```

How to add a new test?

Pick the right file, or add a new one (`test_`*`something`*`.py`)



```
tests
├── conftest.py
├── test_async.py
├── test_client.py
├── test_event.py
├── test_server.py
└── test_test_context.py
```

pytest setup

`command_inout_asynch` tests

`DeviceProxy` tests (mostly using TangoTest device)

Event subscription tests

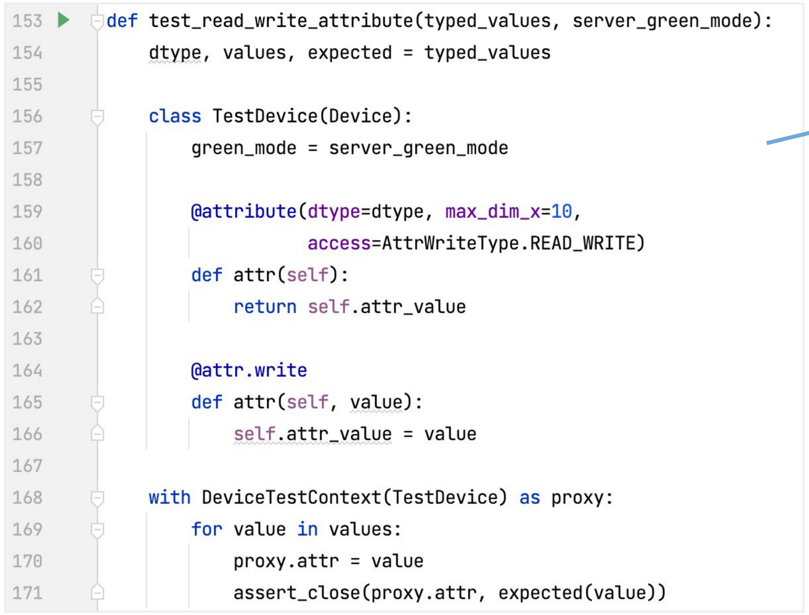
`Device` tests

`DeviceTestContext` tests

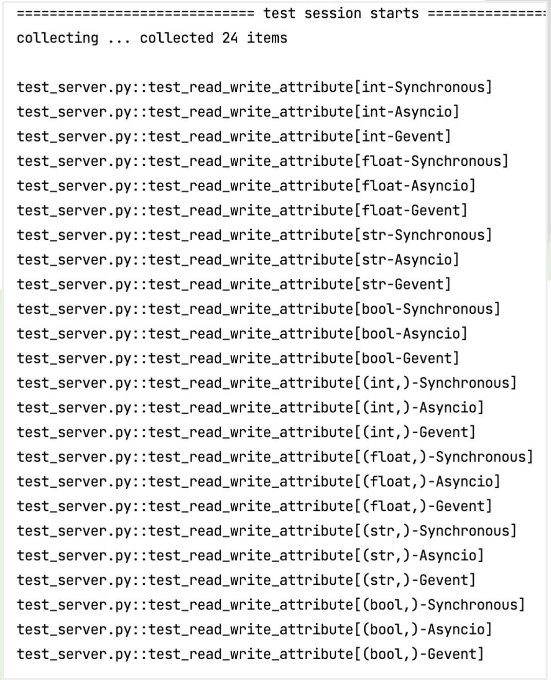
How to add a new test?

Find a similar test and copy the pattern (keep related tests together)

Use existing fixtures to cover many variants easily

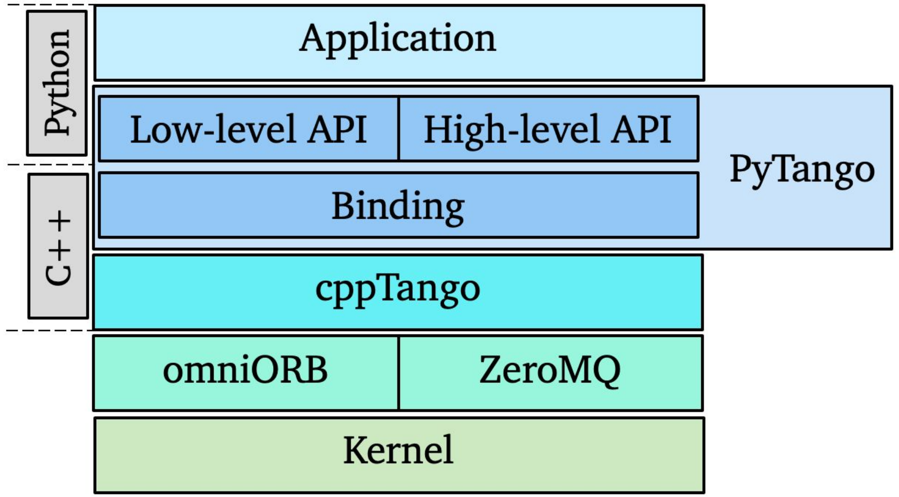


```python
def test_read_write_attribute(typed_values, server_green_mode):
    dtype, values, expected = typed_values

    class TestDevice(Device):
        green_mode = server_green_mode

        @attribute(dtype=dtype, max_dim_x=10,
                   access=AttrWriteType.READ_WRITE)
        def attr(self):
            return self.attr_value

        @attr.write
        def attr(self, value):
            self.attr_value = value

    with DeviceTestContext(TestDevice) as proxy:
        for value in values:
            proxy.attr = value
            assert_close(proxy.attr, expected(value))
```



```
============================ test session starts ==============
collecting ... collected 24 items

test_server.py::test_read_write_attribute[int-Synchronous]
test_server.py::test_read_write_attribute[int-Asyncio]
test_server.py::test_read_write_attribute[int-Gevent]
test_server.py::test_read_write_attribute[float-Synchronous]
test_server.py::test_read_write_attribute[float-Asyncio]
test_server.py::test_read_write_attribute[float-Gevent]
test_server.py::test_read_write_attribute[str-Synchronous]
test_server.py::test_read_write_attribute[str-Asyncio]
test_server.py::test_read_write_attribute[str-Gevent]
test_server.py::test_read_write_attribute[bool-Synchronous]
test_server.py::test_read_write_attribute[bool-Asyncio]
test_server.py::test_read_write_attribute[bool-Gevent]
test_server.py::test_read_write_attribute[(int,)-Synchronous]
test_server.py::test_read_write_attribute[(int,)-Asyncio]
test_server.py::test_read_write_attribute[(int,)-Gevent]
test_server.py::test_read_write_attribute[(float,)-Synchronous]
test_server.py::test_read_write_attribute[(float,)-Asyncio]
test_server.py::test_read_write_attribute[(float,)-Gevent]
test_server.py::test_read_write_attribute[(str,)-Synchronous]
test_server.py::test_read_write_attribute[(str,)-Asyncio]
test_server.py::test_read_write_attribute[(str,)-Gevent]
test_server.py::test_read_write_attribute[(bool,)-Synchronous]
test_server.py::test_read_write_attribute[(bool,)-Asyncio]
test_server.py::test_read_write_attribute[(bool,)-Gevent]
```

Architecture overview



Example of DeviceProxy call to Tango state

Python

```
DeviceProxy.state = green(__DeviceProxy__state)

def __DeviceProxy__state(self, *args, **kwargs):
    """state(self) -> DevState

        A method which returns the state of the device.

      Parameters : None
      Return     : (DevState) constant
"""
    return self._state(*args, **kwargs)
```

Boost defintion

```
void export_device_proxy()
{
    .def("_state", &PyDeviceProxy::state ( arg_("self") ) )
    ....
}
```

Binding to Tango c++

```
namespace PyDeviceProxy
{
static inline Tango::DevState
state(Tango::DeviceProxy& self)
{
       AutoPythonAllowThreads guard;
       return self.state();
}
}
```

Python

```
DeviceProxy.state = green(__DeviceProxy__state)

def __DeviceProxy__state(self, *args, **kwargs):
  """state(self) -> DevState

        A method which returns the state of the device.

      Parameters : None
      Return     : (DevState) constant
"""
  return self._state(*args, **kwargs)
```

The same DeviceProxy call to Tango state but with pybind11 binding which will be the topic of a future webinar

Pybind11 binding

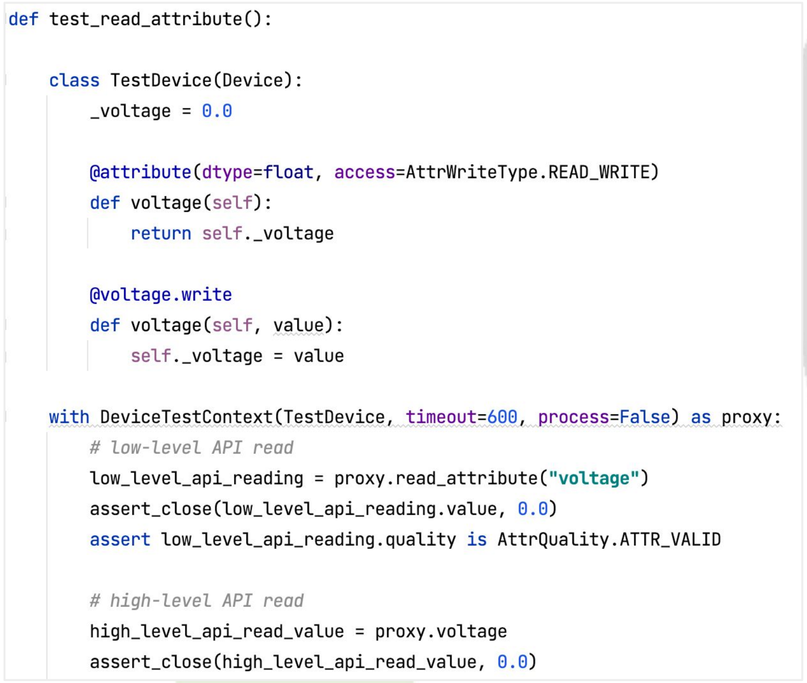
```
void export_device_proxy(py::module &m) {

 .def("_state", [](Tango::DeviceProxy& self) -> Tango::DevState {
          AutoPythonAllowThreads guard;
          return self.state(); // Tango C++ signature
     })
}
```

Practical example: Code navigation.

What happens when an attribute is read?

Client side: `DeviceProxy`

Server side: `Device`



```python
def test_read_attribute():

    class TestDevice(Device):
        _voltage = 0.0

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

        @voltage.write
        def voltage(self, value):
            self._voltage = value

    with DeviceTestContext(TestDevice, timeout=600, process=False) as proxy:
        # low-level API read
        low_level_api_reading = proxy.read_attribute("voltage")
        assert_close(low_level_api_reading.value, 0.0)
        assert low_level_api_reading.quality is AttrQuality.ATTR_VALID

        # high-level API read
        high_level_api_read_value = proxy.voltage
        assert_close(high_level_api_read_value, 0.0)
```

Useful tips - compiling the extension

Compiling the extension

- C++ files in `ext/` create `_tango` shared library
  Example: `build/lib.linux-x86_64-3.8/tango/_tango.cpython-38-x86_64-linux-gnu.so`
- Triggered by `python setup.py build` , `pip install` , etc.

Environments vars used by compilation (in `setup.py`):

- `TANGO_ROOT, OMNI_ROOT, ZMQ_ROOT, BOOST_ROOT`
- Point to installation folders of these packages, e.g., we use `$CONDA_PREFIX` in CI.
- Boost can be tweaked more: `BOOST_HEADERS, BOOST_LIBRARIES, BOOST_PYTHON_LIB`

Useful tips - compilation shortcuts

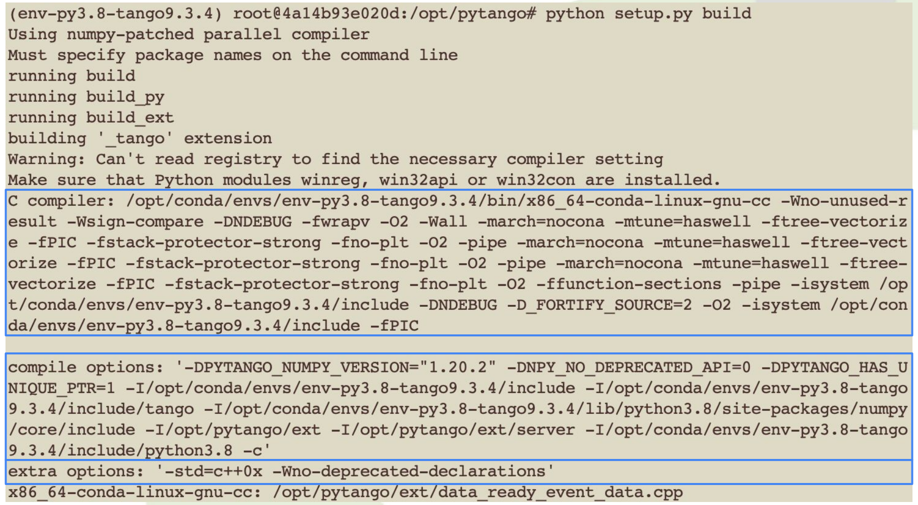
If the `ext/` files haven't changed, and `_tango` file exists can skip compilation:

```
touch build/lib.linux-x86_64-3.8/tango/_tango.cpython-38-x86_64-linux-gnu.so
```

`Makefile`? Not used. Is it up to date? Is being used for pybind11 work.

If working on .cpp file in the extension code, full compilation is slow. Shortcut:

- Compile the single .cpp file
  - Use command from previous build
  - Specify .cpp file after the `-c` option
  - Add `-o` with name of .o file.
- Link the `_tango` file again



```
(env-py3.8-tango9.3.4) root@4a14b93e020d:/opt/pytango# python setup.py build
Using numpy-patched parallel compiler
Must specify package names on the command line
running build
running build_py
running build_ext
building '_tango' extension
Warning: Can't read registry to find the necessary compiler setting
Make sure that Python modules winreg, win32api or win32con are installed.
C compiler: /opt/conda/envs/env-py3.8-tango9.3.4/bin/x86_64-conda-linux-gnu-cc -Wno-unused-r
esult -Wsign-compare -DNDEBUG -fwrapv -O2 -Wall -march=nocona -mtune=haswell -ftree-vectoriz
e -fPIC -fstack-protector-strong -fno-plt -O2 -pipe -march=nocona -mtune=haswell -ftree-vect
orize -fPIC -fstack-protector-strong -fno-plt -O2 -pipe -march=nocona -mtune=haswell -ftree-
vectorize -fPIC -fstack-protector-strong -fno-plt -O2 -ffunction-sections -pipe -isystem /op
t/conda/envs/env-py3.8-tango9.3.4/include -DNDEBUG -D_FORTIFY_SOURCE=2 -O2 -isystem /opt/con
da/envs/env-py3.8-tango9.3.4/include -fPIC

compile options: '-DPYTANGO_NUMPY_VERSION="1.20.2" -DNPY_NO_DEPRECATED_API=0 -DPYTANGO_HAS_U
NIQUE_PTR=1 -I/opt/conda/envs/env-py3.8-tango9.3.4/include -I/opt/conda/envs/env-py3.8-tango
9.3.4/include/tango -I/opt/conda/envs/env-py3.8-tango9.3.4/lib/python3.8/site-packages/numpy
/core/include -I/opt/pytango/ext -I/opt/pytango/ext/server -I/opt/conda/envs/env-py3.8-tango
9.3.4/include/python3.8 -c'
extra options: '-std=c++0x -Wno-deprecated-declarations'
x86_64-conda-linux-gnu-cc: /opt/pytango/ext/data_ready_event_data.cpp
```

Useful tips - crash reports

Run a new Docker image (or from your own environment)

```
docker run --rm -ti -v $PWD:/opt/pytango continuumio/miniconda3:4.9.2 bash
```

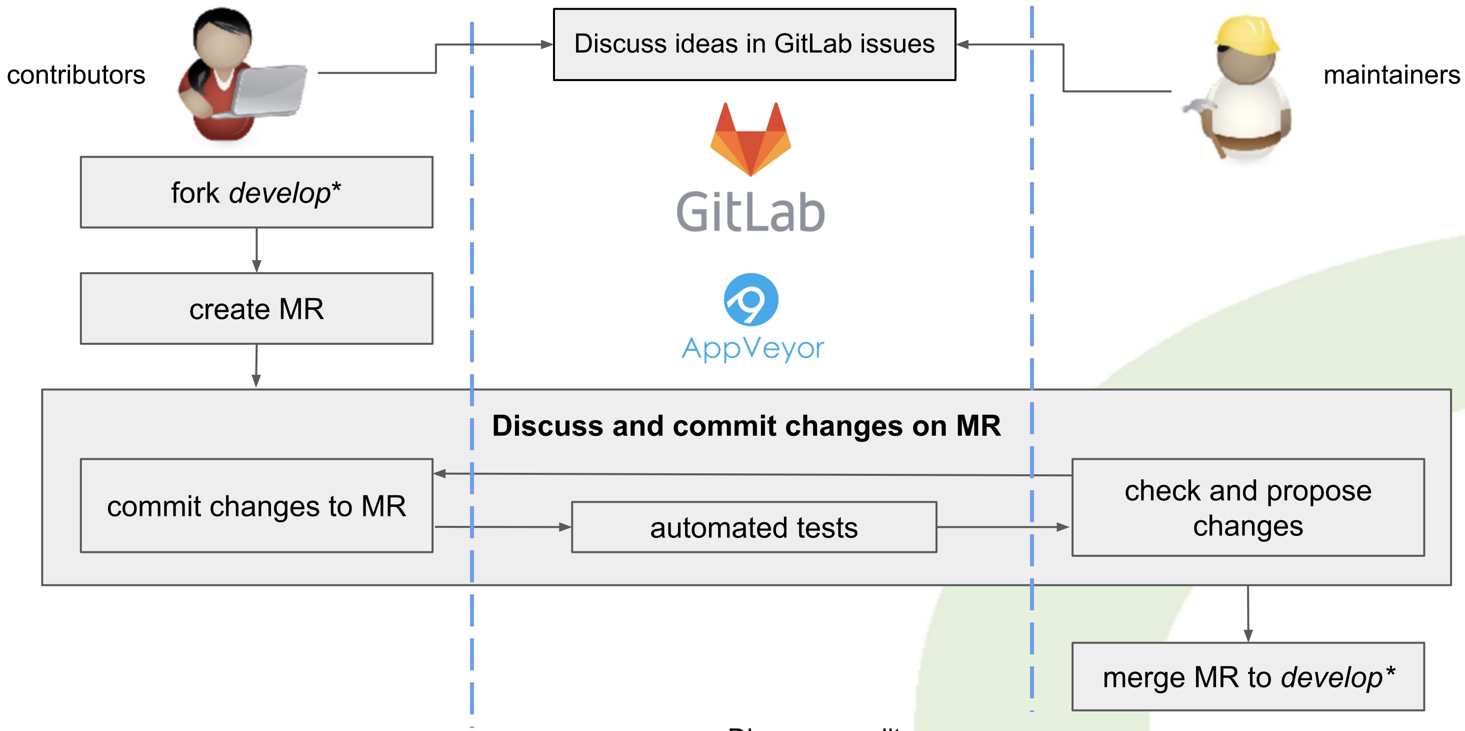
Install PyTango, cppTango+debug symbols, GDB (if necessary)

```
conda create --yes --name env --python=3.8 && conda activate env
conda install -c conda-forge pytango cpptango-dbg
apt update && apt install gdb
```

Run the script that crashes through GDB

```
gdb --args python /opt/pytango/my_script.py
(gdb) run
...
Thread 1 "python" received signal SIGSEGV, Segmentation fault.
0x00007f97c550cb78 in Tango::EventConsumer::unsubscribe_event
(this=0x56450a640a20, event_id=1)
    at /usr/local/src/conda/cpptango-9.3.4/cppapi/client/event.cpp:2028
(gdb) bt
```

Contribution workflow



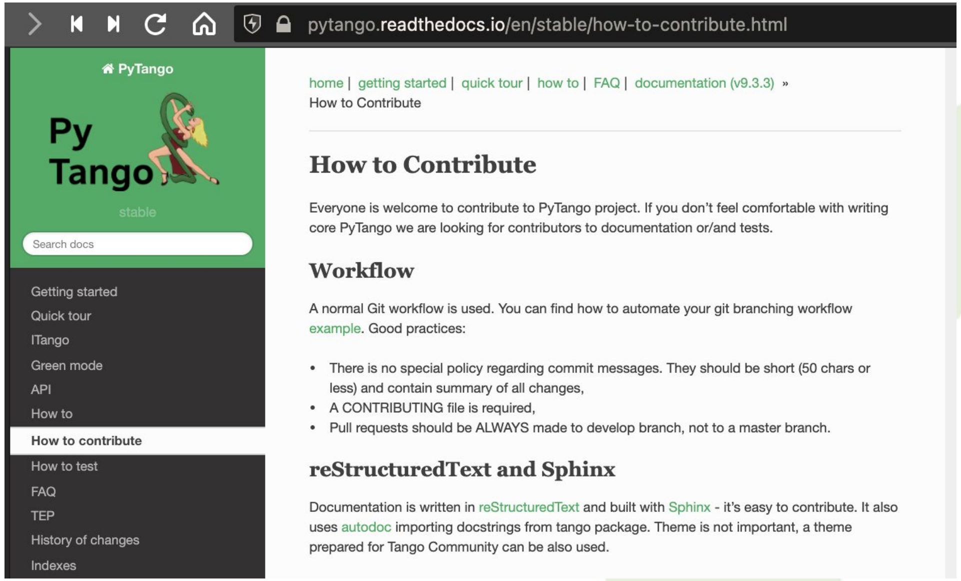
* *develop* branch to be renamed *main*, to match cpptango

Diagram credit:
https://github.com/sardana-org/sardana-training/tree/master/developers

Contribution workflow

More details in the online docs:

https://pytango.readthedocs.io/en/stable/how-to-contribute.html



How to Contribute

Everyone is welcome to contribute to PyTango project. If you don't feel comfortable with writing core PyTango we are looking for contributors to documentation or/and tests.

Workflow

A normal Git workflow is used. You can find how to automate your git branching workflow example. Good practices:

- There is no special policy regarding commit messages. They should be short (50 chars or less) and contain summary of all changes,
- A CONTRIBUTING file is required,
- Pull requests should be ALWAYS made to develop branch, not to a master branch.

reStructuredText and Sphinx

Documentation is written in reStructuredText and built with Sphinx - it's easy to contribute. It also uses autodoc importing docstrings from tango package. Theme is not important, a theme prepared for Tango Community can be also used.

TANGO

Connecting things together

Thank you!

Any questions?

https://gitlab.com/tango-controls/pytango

https://www.tango-controls.org