PyTango and Fandango Workshop

Anton Joubert (SARAO) - Sergi Rubio Manrique (ALBA)

ICALEPCS 2019 - New York

*

GitHub: ajoubertza/icalepcs-workshop

Slides: https://ajoubertza.github.io/icalepcs-workshop

Acknowledgements

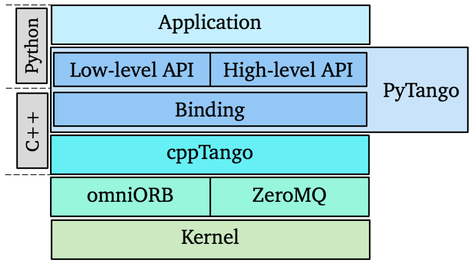
Some of the content of this presentation is from work by:

- Vincent Michel
- Tiago Coutinho
- Antoine Dupré

Thanks!

What is PyTango?

- 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



What is PyTango?

… plus some extras:

- Pythonic API
- asyncio and gevent event loop
- ITango (a separate project)
- Experimental TANGO Database server (sqlite backend)

What's on the menu?

- ITango, a powerful client interface
- Writing tango servers with 15 lines of python
- Testing our servers without a database
- New features being considered
- Fandango - the Swiss army knife

What's on the menu?

Requirements for this workshop:

- TANGO Box VM
- A tiny bit of Python knowledge

ITango

Features

- IPython (jupyter) console
- Direct access to tango classes
- TANGO class sensitive device name auto-completion
- Event monitor
- Qt console
- Notebook
- User friendly error handling

ITango

Hands on

```
is not
```

ITango

ITango

Built-in event monitor - magic command

```
               is
```

Run for more info

ITango

End of ITango demo

- Lots more info on this page: pythonhosted.org/itango
- And don't forget it can be used from a Jupyter notebook

Wow! Writing device servers has never been so easy!

Device servers with pytango >=9.2.1

```
from      import
from                import

class PowerSupply


    def voltage
        return


    def calibrate


if
```

See file:

Testing time!

Server:

Client:

```
import
```

Let's try out events!

Adding a polled attribute - see file:

```
import







    def random
        return
```

Going back to ipython:

```

```

Enumerated types

Add an enumerated type - see file:

```
import


class TrackingMode







    def output_tracking
        return
```

```










         False
```

Unit testing

```
from       import
from                  import

from     import


def test_calibrate

    with                                        True  as

        assert
```

See file:

launches tango device server in a subprocess, and returns a instance connected to it. No DB, so limited functionality.

"Sort-of" unit testing - can test from client's perspective, but cannot access device's methods or attributes directly.

Unit testing

Events are tricky - may need to provide port number too

```
def test_events


    def callback
        if not



    with                                            True             as



        assert
```

See file:

Unit testing

```
                                                        in
```

Unit testing

is the default

If starting device more than once, probably get segmentation fault.

Options:

-
- nosetest can use plugin:
- pytest can use plugin:

Asynchronous pytango

Also called green modes, checkout the docs:

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

Asyncio client mode example

```
from                   import                 as

         await

         await
```

A simple TCP server for tango attributes

- Try this simple TCP server for Tango attributes
- It runs on all interfaces on port 8888:

```
```

- It can be accessed through netcat:

```
```

More resources

Asyncio overview

- Slides: vxgmichel.github.io/asyncio-overview
- Repo: github.com/vxgmichel/asyncio-overview

Previous PyTango workshop (notes on concurrency)

ICALECPS 2017

- Slides: vxgmichel.github.io/icalepcs-workshop
- Repo: github.com/vxgmichel/icalepcs-workshop

New features being considered

1. Python logging as standard, sends to TANGO Logging Service (bringing in feature from fandango)

Option 1 - *Opt-in*: mixin adds method and object

```
class PowerSupply

    def calibrate
```

Option 2 - *Opt-out*: part of , disable via overriding

```
class PowerSupply
```

User could add/remove handlers, e.g., syslog or Elastic instead of TLS.

New features being considered

2. Support forwarded attributes with `DeviceTestContext`

Currently problem with missing root attribute

3. `faketango.Device` for basic unit testing:

```
import
from       import
from                  import
from                  import

from                         import


def test_init



    assert
```

(This may be difficult, and have limitations - polling, events, green modes, ...)

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 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
- Fork the repo, make it better, make a PR. Thanks!
- More info in how-to-contribute.

PyTango versions

- PyPI has the latest
    - but binding extension not compiled for Linux
    - binding is compiled and statically linked for Windows
- Linux packages
    - The binding is already compiled code, so quick to install.
    - Typically a few versions behind.

Done! Any questions?

Fandango - a Swiss army knife for tango

ICALEPCS 2019 - New York Sergi Rubio ManriqueICALEPCS 2019 - New York

ICALEPCS 2019 - New York

What is Fandango?

- a Python library: pip install fandango
- and a shell script: fandango read_attribute test/dyn/1/t
- https://github.com/tango-controls/fandango
- uses PyTango and DatabaseDS and Starter Device Servers

What is Fandango?

It originated from 2 motivations:

- provide a library with utilities/templates for PyTango devices at ALBA
- the desire to get completely rid of Java applications (Jive and Astor)

What is Fandango?

It provides many features:

- the origin, functional programming for tango (fun4tango)
- features from Java clients (Jive, Astor)
- utilities for python devices (Logging, Threading, Workers)
- includes methods for functional programming
- enables middle-layer devices (DynamicDS, SimulatorDS, CopyCatDS)

fandango submodules

- functional: functional programming, data format conversions, caseless regular expressions
- tango : tango api helper methods, search/modify using regular expressions
- dynamic : dynamic attributes, online python code evaluation
- server : Astor-like python API
- device : some templates for Tango device servers
- interface: device server inheritance
- db: MySQL access
- dicts,arrays: advanced containers, sorted/caseless list/dictionaries, .csv parsing
- log: logging
- objects: object templates, singletones, structs
- threads: serialized hardware access, multiprocessing
- linos: accessing the operative system from device servers
- web: html parsing
- qt: some custom Qt classes, including worker-like threads.

fandango.tango submodules

- command: asynchronous execution of tango commands on a background thread
- eval/tangoeval: evaluation of formulas using tango attribute values
- dynattr: dynamic typing of attributes, used to override operators on demand
- export: import/export tango attributes/devices/properties on json/pickle formats
- search: methods to search devices/attributes in the tango database or a running control system
- methods: miscellaneous methods to access Tango devices and attributes

fandango vs PyTango

PyTango is a binding of TANGO C++, thus bringing the same functionality and mimicking the same methods and arguments available on C++.

The PyTango High Level API provides a pythonic API for developing TANGO device servers and clients in Python 3.

fandango instead, extends the API adding some features only available on Java clients like Jive and Astor, the default management UI applications of TANGO.

fandango vs PyTango

Adding a new device with *PyTango* (mimics the C++ API):

```
                              None

                                                               and class
        are specified in the DbDevInfo structure

        Example
```

fandango vs PyTango

Adding a new device with *fandango* (mimics the Jive UI form):

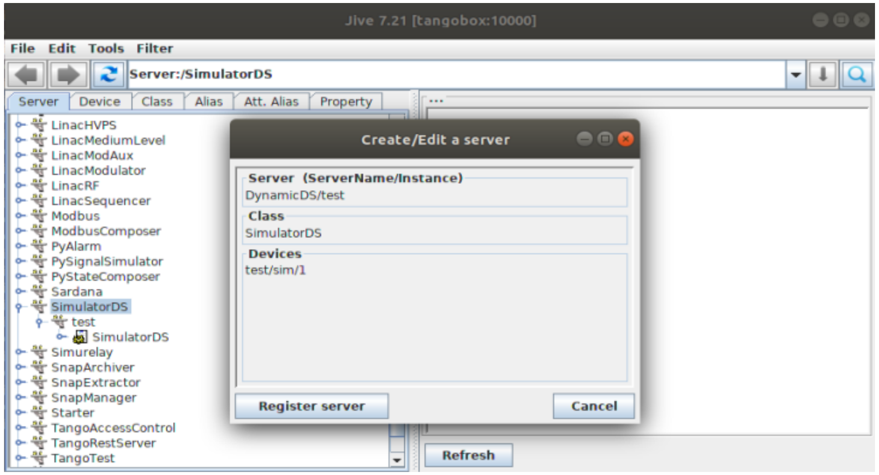
fandango.tango: creating and launching devices

fandango provides Astor python API, providing the same functionality than astor tool.

fandango can be used in python:

```
import          as
```

fandango.tango: creating and launching devices



Server (ServerName/Instance)
DynamicDS/test

Class
SimulatorDS

Devices
test/sim/1

fandango.tango: creating and launching devices

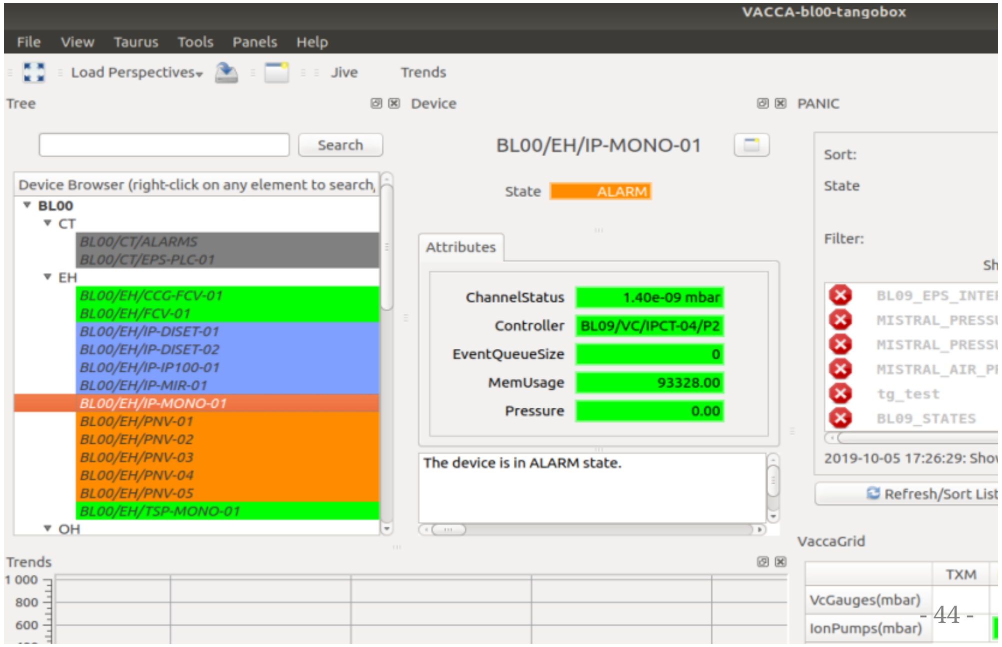
methods from fandango can also be launched linux shell:

fandango.tango: creating and launching devices

fandango.tango: searching in the database

fandango.tango: searching in the database

Import/Export Device servers from TANGO Db



BL00
- CT
  - BL00/CT/ALARMS
  - BL00/CT/EPS-PLC-01
- EH
  - BL00/EH/CCG-FCV-01
  - BL00/EH/FCV-01
  - BL00/EH/IP-DISET-01
  - BL00/EH/IP-DISET-02
  - BL00/EH/IP-IP100-01
  - BL00/EH/IP-MIR-01
  - BL00/EH/IP-MONO-01
  - BL00/EH/PNV-01
  - BL00/EH/PNV-02
  - BL00/EH/PNV-03
  - BL00/EH/PNV-04
  - BL00/EH/PNV-05
  - BL00/EH/TSP-MONO-01
- OH

BL00/EH/IP-MONO-01

State: ALARM

ChannelStatus	1.40e-09 mbar
Controller	BL09/VC/IPCT-04/P2
EventQueueSize	0
MemUsage	93328.00
Pressure	0.00

The device is in ALARM state.

- BL09_EPS_INTE
- MISTRAL_PRESS
- MISTRAL_PRESS
- MISTRAL_AIR_P
- tg_test
- BL09_STATES

2019-10-05 17:26:29: Sho

	TXM
VcGauges(mbar)	
IonPumps(mbar)	

Import/Export Device servers from TANGO Db

Exporting/Importing devices and properties declaration allows to easily create/move hundreds of devices with a few commands:

Import/Export Device servers from TANGO Db

```
import          as


                                     for   in
```

Import/Export Device servers from TANGO Db

Import/Export Device servers from TANGO Db

although csv is less popular, tango2csv allows human-readable exports

Evaluating attribute values on runtime

fandango provides two implementations for evaluating python code for attributes:

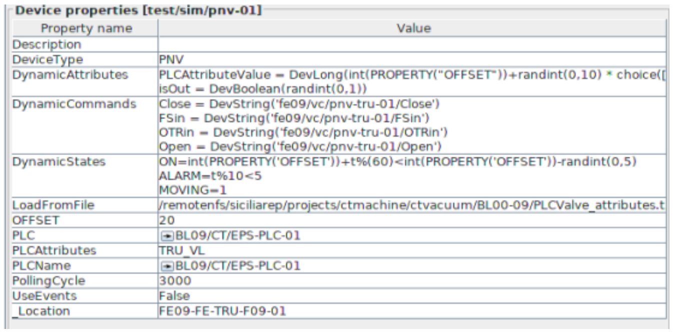
- DynamicDS: device template for creating attributes dynamically using properties, optimized for reading hundreds of attributes, implementing caches and hierarchic evaluation.
- TangoEval: generic python evaluator object with Tango syntax parsing, it can be used from either devices or clients

Evaluating attribute values on runtime

Declaring Dynamic Attributes on a simulator/composer/processor device:

```
                                                                                   (
                                                                      or       for
```

Evaluating attribute values on runtime



Device properties [test/sim/pnv-01]

Property name	Value
Description	
DeviceType	PNV
DynamicAttributes	PLCAttributeValue = DevLong(int(PROPERTY("OFFSET"))+randint(0,10) * choice([isOut = DevBoolean(randint(0,1))
DynamicCommands	Close = DevString('fe09/vc/pnv-tru-01/Close') FSin = DevString('fe09/vc/pnv-tru-01/FSin') OTRin = DevString('fe09/vc/pnv-tru-01/OTRin') Open = DevString('fe09/vc/pnv-tru-01/Open')
DynamicStates	ON=int(PROPERTY('OFFSET'))+t%(60)<int(PROPERTY('OFFSET'))-randint(0,5) ALARM=t%10<5 MOVING=1
LoadFromFile	/remotenfs/siciliarep/projects/ctmachine/ctvacuum/BL00-09/PLCValve_attributes.t
OFFSET	20
PLC	BL09/CT/EPS-PLC-01
PLCAttributes	TRU_VL
PLCName	BL09/CT/EPS-PLC-01
PollingCycle	3000
UseEvents	False
_Location	FE09-FE-TRU-F09-01

Evaluating attribute values on runtime

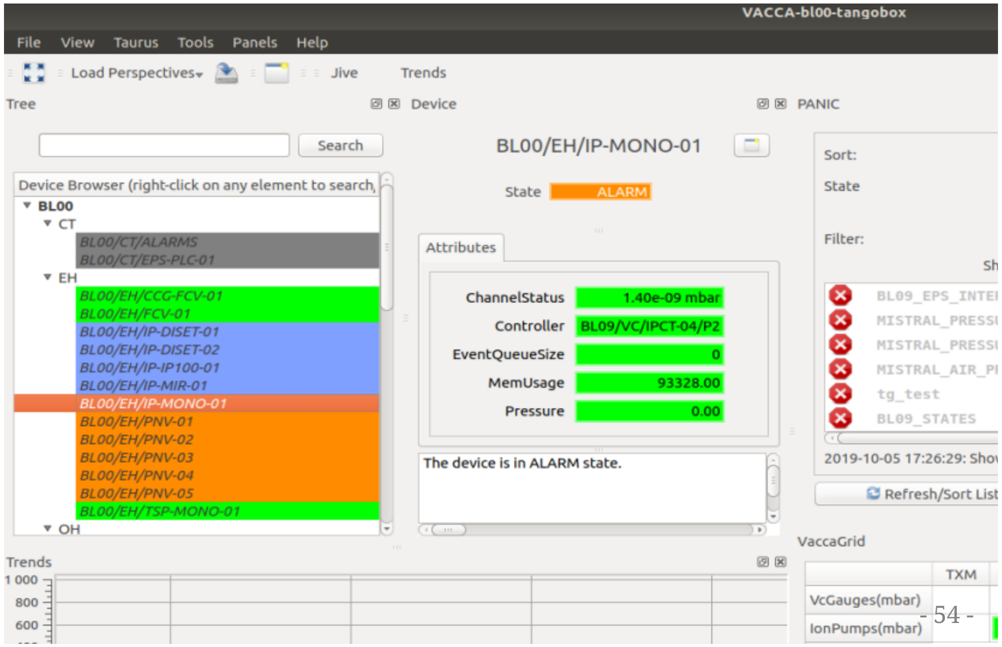
Declaring a formula in the PANIC Alarm System (using fandango.TangoEval):

```


                                           or


             or        in                        for   in








                 for   in
```

Libraries/Projects using fandango

- SimulatorDS Device Server
- CopyCatDS, ComposerDS, PyStateComposer, PyAttributeProcessor, ...
- PANIC Alarm System: [https://github.com/tango-controls/panic]
- PyTangoArchiving
- PyPLC Device Server
- VacuumController Device Servers (Varian, Agilent, MKS, Pfeiffer)
- VACCA User Interface

Fandango and VACCA



Device Browser (right-click on any element to search,

- BL00
  - CT
    - BL00/CT/ALARMS
    - BL00/CT/EPS-PLC-01
  - EH
    - BL00/EH/CCG-FCV-01
    - BL00/EH/FCV-01
    - BL00/EH/IP-DISET-01
    - BL00/EH/IP-DISET-02
    - BL00/EH/IP-IP100-01
    - BL00/EH/IP-MIR-01
    - BL00/EH/IP-MONO-01
    - BL00/EH/PNV-01
    - BL00/EH/PNV-02
    - BL00/EH/PNV-03
    - BL00/EH/PNV-04
    - BL00/EH/PNV-05
    - BL00/EH/TSP-MONO-01
  - OH

BL00/EH/IP-MONO-01

State ALARM

Attributes

ChannelStatus	1.40e-09 mbar
Controller	BL09/VC/IPCT-04/P2
EventQueueSize	0
MemUsage	93328.00
Pressure	0.00

The device is in ALARM state.

- BL09_EPS_INTE
- MISTRAL_PRESSU
- MISTRAL_PRESSU
- MISTRAL_AIR_PI
- tg_test
- BL09_STATES

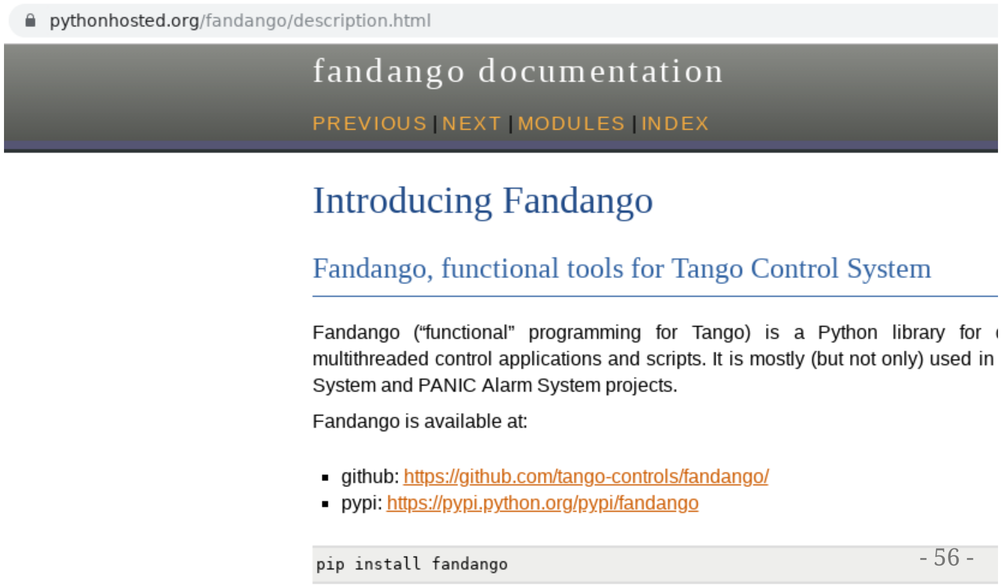
2019-10-05 17:26:29: Show

	TXM
VcGauges(mbar)	
IonPumps(mbar)	

Plenty of useful methods:

Fandango documentation

https://pythonhosted.org/fandango


pythonhosted.org/fandango/description.html
fandango documentation
PREVIOUS | NEXT | MODULES | INDEX

Introducing Fandango

Fandango, functional tools for Tango Control System

Fandango ("functional" programming for Tango) is a Python library for
multithreaded control applications and scripts. It is mostly (but not only) used in
System and PANIC Alarm System projects.

Fandango is available at:

- github: https://github.com/tango-controls/fandango/
- pypi: https://pypi.python.org/pypi/fandango

```
pip install fandango
```

What is missing?

The most requested feature:

- PyTango 3

Which is currently blocked by:

- Testing and CI

Two ports to python 3 actually exist (one by me and another from S2Innovation), but none of them has been yet put in production.