TANGO / PyTango Training

Lorenzo Pivetta / Sergio Rubio Manrique

3-6 March 2020, Perth, Australia

Introduction to PyTango

Materials used in this training thanks to :

- Vincent Michel
- Tiago Coutinho
- Antoine Dupré
- Anton Joubert

https://pytango.readthedocs.io/

https://pytango.readthedocs.io/en/stable/quicktour.html

https://pytango.readthedocs.io/en/stable/howto.html

PyTango is a python module that exposes to Python the complete Tango C++ API. This means that you can write not only tango applications (scripts, CLIs, GUIs) that access tango device servers but also tango device servers themselves, all of this in pure python.

PyTango wraps Tango C++ libraries using Boost:Python. A new version is currently under development using pybind11. It will provide a lighter wrapper and a more modern approach.

Using a Python wrapper on top of Tango C++ libraries helps to standardize the behaviour of the two development APIs.

Documentation and feature examples are valid for both languages and it allows to mix Tango Device Servers written in both languages in a single Device Server launcher.

But, using a wrapper on top of C++ force us to use an API that is not pythonic. To solve this issue the Python HL API is provided, allowing the development of fully-pythonic devices.

PyTango API

https://pytango.readthedocs.io/en/stable/server_api/

- API is almost identical than for C++ (all methods available, all documentation and examples are valid)
- It's not pythonic (method names and workflow may feel not natural in python)
- Coding overhead (it's needed to declare many methods that may not be used at all).

PyTango High Level API (HL)

https://pytango.readthedocs.io/en/stable/server_api/server.html

- Use of python primitives like decorators and properties reduce the number of lines of code to implement to less than a half.
- Pythonic, simpler, shorter and clearer code
- Due to API simplification, many functionalities become "hidden" (to use all features, fallback to old API is needed).

Pythonic

Pythonic is an adjective that describes an approach to computer programming that agrees with the founding philosophy of the Python programming language. There are many ways to accomplish the same task in Python, but there is usually one preferred way to do it. This preferred way is called "pythonic."

- Python philosophy of writing code
- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.

Topics not covered on the training

- Jupyter notebooks
- Unit Testing
- TGQL
- Green modes (asyncio) ...

Questions for participants

- Who has experience with python?
- Who has experience with Tango?
- Who has read SKA developer guidelines?
- Who has used already the lmc-base-classes?
- How many read my email?
- How many installed virtualbox?
- How many use Windows?
- How many use Mac OSX?
- How many use Ubuntu?
- How many use other Linux OS?

Setup of the Tango Development Environment

The training will be based on the existing Tango Development Environment for SKA:

https://developer.skatelescope.org/en/latest/tools/tango-devenv-setup.html

This training will use the existing docker containers for database, Tango Database Server and itango CLI.

I recommend you to test the environment setup procedure in advance, so we can detect and solve any problem with your OS/hardware.

I've tested the development docker containers on Ubuntu (works), Debian (doesn't work) and Lubuntu (better performance). So in case your system is other than Ubuntu, it is highly recommended to install virtual box before the training:

https://www.virtualbox.org/

These are the OS images that can be used for the training:

UBUNTU (full-featured):
https://sourceforge.net/projects/osboxes/files/v/vb/55-U-u/18.04/18.04.2/18042.64.7z/download

LUBUNTU (much, much faster):
https://sourceforge.net/projects/osboxes/files/v/vb/33-Lb--t/18.04/18.04.3/L1804.3-64bit.7z/download

Using the Lubuntu (non-standard) image

The lubuntu image would require these two lines to be removed on deploy_tangoenv.yml file:

```
install_ide: 'yes'
- ide
```

And you will have to use lighter editors for the training (mousepad or geany instead of PyCharm). This will not be an impediment for the training as all the exercises can be done using a simple lightweight editor.

Verifying the installation

just do

```
cd /usr/src/ska-docker/docker-compose
make up
make start jive
```

And an application like this should appear on screen:

`https://tango-controls.readthedocs.io/en/9.2.5/_images/jive.jpg`

Setup of itango container to run SKABaseDevice Tango classes

ska_logging and skabase libraries will have to be downloaded:

```
# On the home folder of the VBox user
git clone https://gitlab.com/ska-telescope/ska-logging
git clone https://github.com/ska-telescope/lmc-base-classes
ln -s ska-logging/ska_logging
ln -s lmc-base-classes/skabase
```

then, modify the itango.yml file to mount host home on the container:

```
volumes:
  - ${HOME}:/hosthome:rw
```

this will allow you to launch any SKABaseDevice inheriting device using docker:

```
cd /usr/src/ska-docker/docker-compose
make start itango #not needed if already shows up in "make status"
docker exec -it -e PYTHONPATH=/hosthome python3 /hosthome/YourSKADevice.py
```

Verifying the setup

just run:

```
cd /usr/src/ska-docker/docker-compose
make start itango #not needed if already shows up in "make status"
docker exec -it -e PYTHONPATH=/hosthome itango python3
/hosthome/skabase/SKABaseDevice/SKABaseDevice.py -?
```

and this message will appear on screen (waiting for a server to be declared):

```
usage :  SKABaseDevice instance_name [-v[trace level]] [-nodb [-dlist <device name
list>]]
Instance name defined in database for server SKABaseDevice :
```

Writing a TANGO device class (Python)

Concepts that will be explored:

- Adding commands
- Modifying device state
- Adding attributes
- Attribute alarm ranges
- Implications of polling/events

Writing Python Tango Devices using both classical API and Python HL API

Exercise 1:

Write A Tango Device Server for a Fixed Voltage Power Supply following this specification. (The device is loosely based on the Power Supply device that appears on the PyTango HL documentation).

The Device Server that we are going to create will simulate a Voltage Power Supply. When setting On() the Voltage written by user will be applied on a LoadImpedance, resulting in a Current reading readable by clients.

Properties

- LoadImpedance: DevDouble: in this example it will be the impedance of the load connected to the power supply.
- HWReadtime: Time needed to change a parameter on HW

States

- ON: Output is enabled
- OFF: Output is disabled (although attributes are readable)
- RUNNING: A setting is being applied (will make sense in a later exercise)

Commands

- On : enables the output, current is calcullated taking voltage and load impedance into account.

- Off : disables the output, current is 0

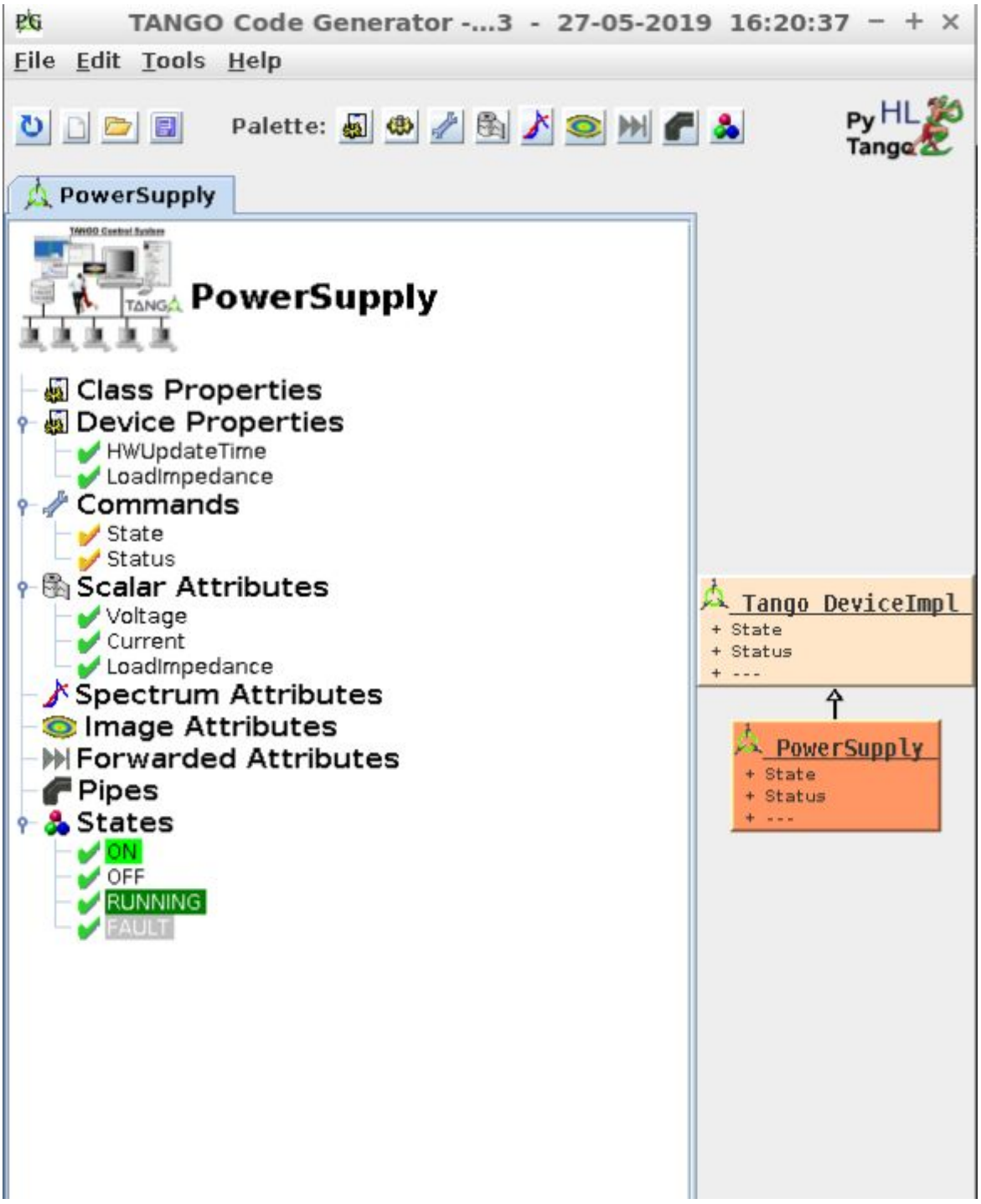
Attributes

- Voltage : writable by user, needs some time to be updated on HW (Units in Volts)
- Current: calcullated by the device when state is ON (Units in Ampers)

Executing Pogo from a container

```
cd /usr/src/ska-docker/docker-compose
make start pogo
```

Design your device using the Pogo Helper Tool



Proceed to create the described Properties, Commands and Attributes.

When done, proceed to create the device in Jive and launch it from shell.

Creating and executing the device

We will launch the Jive application

```
cd /usr/src/ska-docker/docker-compose
```

```
make start jive
```

From there, we will go to "Add new device" in the menu with the following parameters

```
Server: PowerSupply/1
 Class: PowerSupply
 Devices: test/ps/1
```

Now we can launch the device, either locally or using the itango container.

Running the device inside a container

```
cd /usr/src/ska-docker/docker-compose
```

Add these lines to itango.yml

```
# sudo gedit itango.yml

volumes:
  - ${HOME}:/hosthome:rw
```

Then, launch the container and run the device:

```
make start itango
docker exec -it /hosthome/PowerSupply.py -?
```

Running the device in the virtual machine

This option is far more comfortable, but we will not be using nor python3 nor the latest versions of packages (not relevant for the training).

sudo apt-get update sudo apt install python-tango python PowerSupply.py

---

Exercise 2:

Implementation of the Device. Insert the described behaviour in Exercise 1 into the PROTECTED AREAS of the generated template.

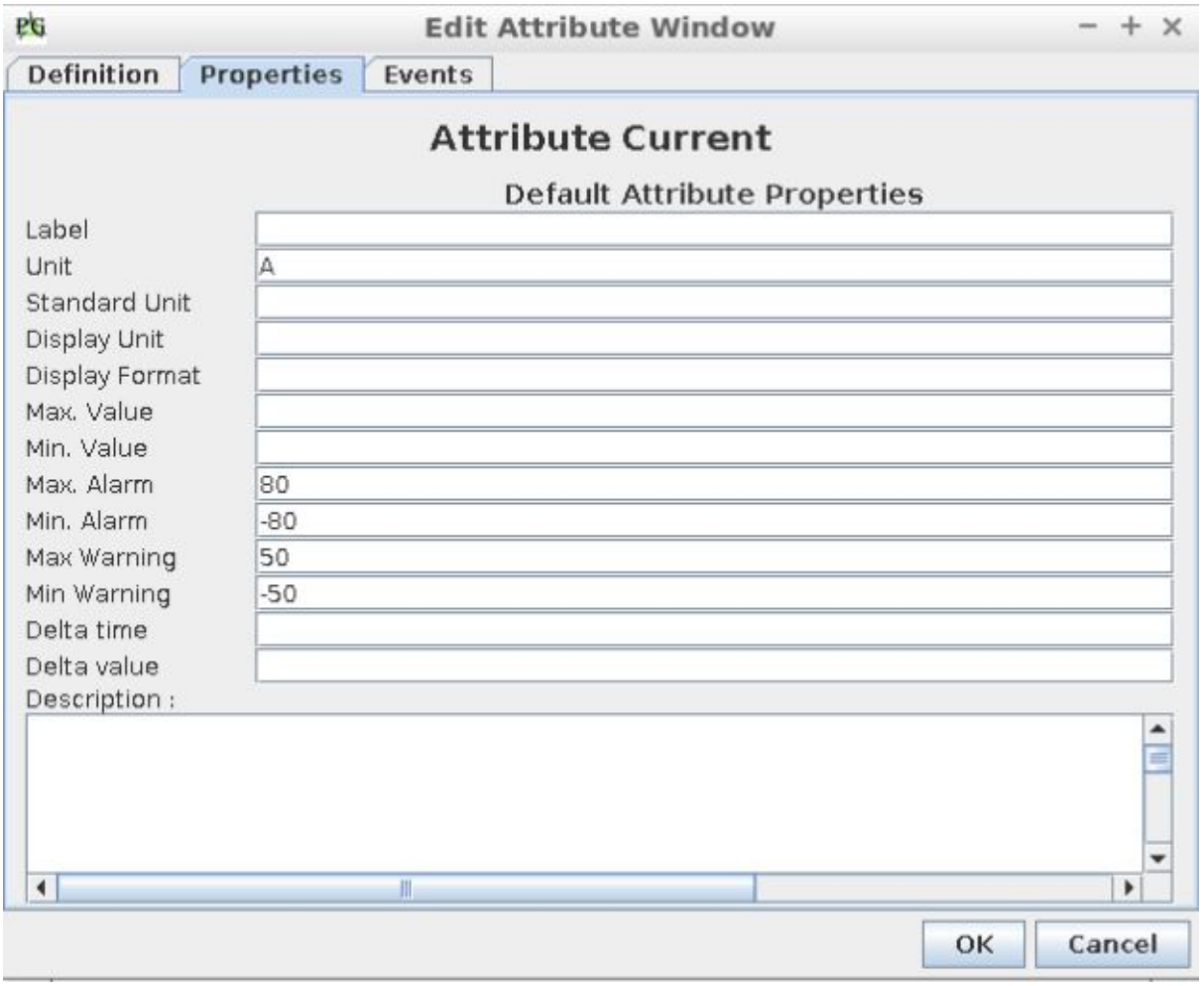
These methods will be useful:

- Modify the state: self.set_state(DevState.ON)
- (you may use ipython to check the valid values of DevState)
- generate an sleep within the code: time.sleep(seconds)

Exercise 3:

Setup polling for State, Units for Current and Voltage, Max and Min valid values for Voltage, Alarm Ranges for Current.

Use the different context menus in pogo to setup state and attributes:



Once everything is setup:

- regenerate the code.
- Check if your previous implementation was destroyed (ups!).
- Execute the following code in a shell:

```
cd /usr/src/ska-docker/docker-compose
docker exec -it itango ipython3
```

Showing the PowerSuplly current changes

Using a generic Event callback

```
import tango
from tango.client import Device
dev = Device('test/ps/1')
cb = tango.utils.EventCallback()

eis = [dev.subscribe_event(a,tango.EventType.CHANGE_EVENT,cb) for a in
('Current','State')]

#... play with your device

[dev.unsubscribe_event(e) for e in eis]
```

Writing your own custom callback

```
dev = Device('test/ps/1')

def callback(event):
    print(event.get_date(),event.device,event.attr_name)
    if event.err:
        print('error received!')
    else:
        print('value',event.attr_value.value)

ei = dev.subscribe_event('Current',tango.EventType.CHANGE_EVENT,callback)

dev.Current = 10

dev.unsubscribe_event(ei)
```

Exercise 4: Playing with the device

- Explore the management of polling/events from Jive.

- Modify the properties, what is needed to update them?
- Increase the HWReadtime ... what is the effect?
- Access the device from ATKPanel and ipython, what is the effect?

"Classic" Python API vs Python HL

Some advantages of "HL" API:

- Much less overhead on writing device servers and clients.
- Python properties and decorators allow a much more compact syntax.
- More readable code allows a better understanding of the behaviour.

Some advantages of "classic" ap:

- Dynamic Attributes are easier to implement, as "attr" object passed as argument allow to identify attributes by name.
- Dynamic Commands implemented only on classic API
- read_attr_hardware exists only on classic API, allows to treat attribute reading and command execution separately.
- (but you can still write your own custom read_attr_hardware if you want)
- separate init device allows to separate creation of variables that should be done only once from those done at each Init() call. (making unnecessary to have a Reset() command).
- (but you can still write your own custom init if you want)

`<< Live Review of the same class, but generated with the "Old API" >>`

Writing a TANGO CLI client (iPython)

- Command line clients
- itango

```
$ itango
ITango 9.3 -- An interactive Tango client.
Running on top of Python 2.7.15, IPython 5.8 and PyTango 9.3
[...]

# `Device` is an alias for `tango.DeviceProxy`
In [1]: dev = Device("sys/tg_test/1")

# or can use class name (limits <tab> search space)
In [2]: dev = TangoTest("sys/tg_test/1")

# is the device running?
In [3]: dev.ping()
API_DeviceNotExported: Device sys/tg_test/1 is not exported
[...]

# No!  Start it, and try again...
In [4]: dev.ping()
Out[4]: 1771
```

```
# send a command (hard way)
In [5]: dev.command_inout('DevShort', 1234)
Out[5]: 1234

# send a command (easy way - PyTango feature)
In [6]: dev.DevShort(1235)
Out[6]: 1235

# read an attribute
In [7]: dev.long_spectrum
Out[7]:
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[...]
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)

# write to it
In [8]: dev.long_spectrum = (1, 2, 3, 4)

In [9]: dev.long_spectrum
```

```
Out[9]: array([1, 2, 3, 4], dtype=int32)
```

Built-in event monitor - magic `mon` command

```
In [11]: dev.poll_attribute('State', 1000)

In [10]: mon -a sys/tg_test/1/State
'sys/tg_test/1/State' is now being monitored. Type 'mon' to see all events

In [7]: dev.SwitchStates()

In [9]: mon
ID   Device              Attribute    Value       Quality    Time
---- ------------------- ------------ ----------- ---------- ---------------
   0 sys/tg_test/1       state        RUNNING     ATTR_VALID  16:52:28.564090
   1 sys/tg_test/1       state        RUNNING     ATTR_VALID  16:52:29.564329
   2 sys/tg_test/1       state        FAULT       ATTR_VALID  16:52:41.564279

In [10]: mon -d sys/tg_test/1/State
Stopped monitoring 'sys/tg_test/1/State'
```

Run `mon?` for more info <<< Doesnt work on itango3!

Showing the PowerSuplly current changes

```
dev = Device('test/fixedcurrps/1')

def callback(event):
    print(event.get_date(),event.device,event.attr_name)
    if event.err:
        print('error received!')
    else:
        print('value',event.attr_value.value)

ei = dev.subscribe_event('Current',tango.EventType.CHANGE_EVENT,callback)

dev.Current = 10

dev.unsubscribe_event(ei)


dev = Device('test/fixedcurrps/1')
cb = tango.utils.EventCallback()

eis = [dev.subscribe_event(a,tango.EventType.CHANGE_EVENT,cb) for a in
('Current','State')]
```

```
dev.Current = 15

list(map(dev.unsubscribe_event,eis))
```

End of ITango hands-on

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

Polling from Clients vs Polling from Server vs Pushing Events

- A completely lazy implementation will consist on letting the client to ask periodically the values it wants.
    - Advantages, nothing is done if nobody wants it
    - Disadvantage, many clients will easily saturate the device
- Using Tango Polling, clients only access cached values or receive events
    - Advantages, most efficient, you protect the device from external clients
    - Disadvantages, the device is running continuously, any memory leak will be magnified, the device becomes sensible to serialization issues (if not giving time to process all pollings)
- Pushing events manually from the code
    - Typically updating attribute values from a single command execution (Update()) or a background thread.
    - It can be combined with the previous approach
    - Not as easily configurable from Jive (if used, it may introduce performance issues)
    - Even if using a background thread, it does not scape serialization completely (event pushing is blocking)

Pushing events from Class (device) code

To configure an attribute to be pushed:

- On init_device:

```
self.set_change_event('Current',True,False) #Push manually, ignore Jive config
```

- On your methods:

```
self.push_change_event('Current',value) # [timestamp, quality]
```

Exercise 5

Modify your existing device server to push Current events manually instead of using polling

Exercise 6

Add a RampingSteps property and this method to your device to Ramp the current values:

```
   def ramp_current(self,start, end):
       step = float(end-start)/self.RampingSteps
       self.set_state(DevState.RUNNING)
       for i in range(self.RampingSteps):
           time.sleep(1.)
           self.current += step
           self.push_change_event('Current',self.current)
           print(self.current)
       self.set_state(DevState.ON)
```

Observe the Current evolution from ipython / itango

Review of SKA guidelines for TANGO devices

SKABaseDevice Class

All Tango Device Servers in SKA projects are expected to inherit from the SKABaseDevice parent class.

This class depends on two models that must be divided in the PYTHONPATH:

- ska_logging: https://gitlab.com/ska-telescope/ska-logging
- skabase: https://github.com/ska-telescope/lmc-base-classes

This template device overrides Tango logging mechanims to use python logging instead.

So, instead of the 4 basic loggins in Tango managed by shell argument (-v4) SKA devices will display the following logs:

```
OFF, FATAL, ERROR, WARNING, INFO, DEBUG
```

The LoggingLevel attribute and LoggingLevelDefault will control the current output of the logging system.

It also adds several attributes, initializing them using enumerations defined within SKA project:

```
        self._version_id = release.version
         self._health_state = HealthState.OK
         self._admin_mode = AdminMode.ONLINE
         self._control_mode = ControlMode.REMOTE
         self._simulation_mode = SimulationMode.FALSE
         self._test_mode = TestMode.NONE
```

In addition, a Reset() command should allow to reinitialize the device without calling Init() command (that may reset several internal variables unintentionally).

Reusing classes that do not belong to SKA project

Example, using the PySocket class from
https://github.com/alba-synchrotron/PySocket

It can be added to an existing device by modifying the inheritance within the main method of the server launcher.

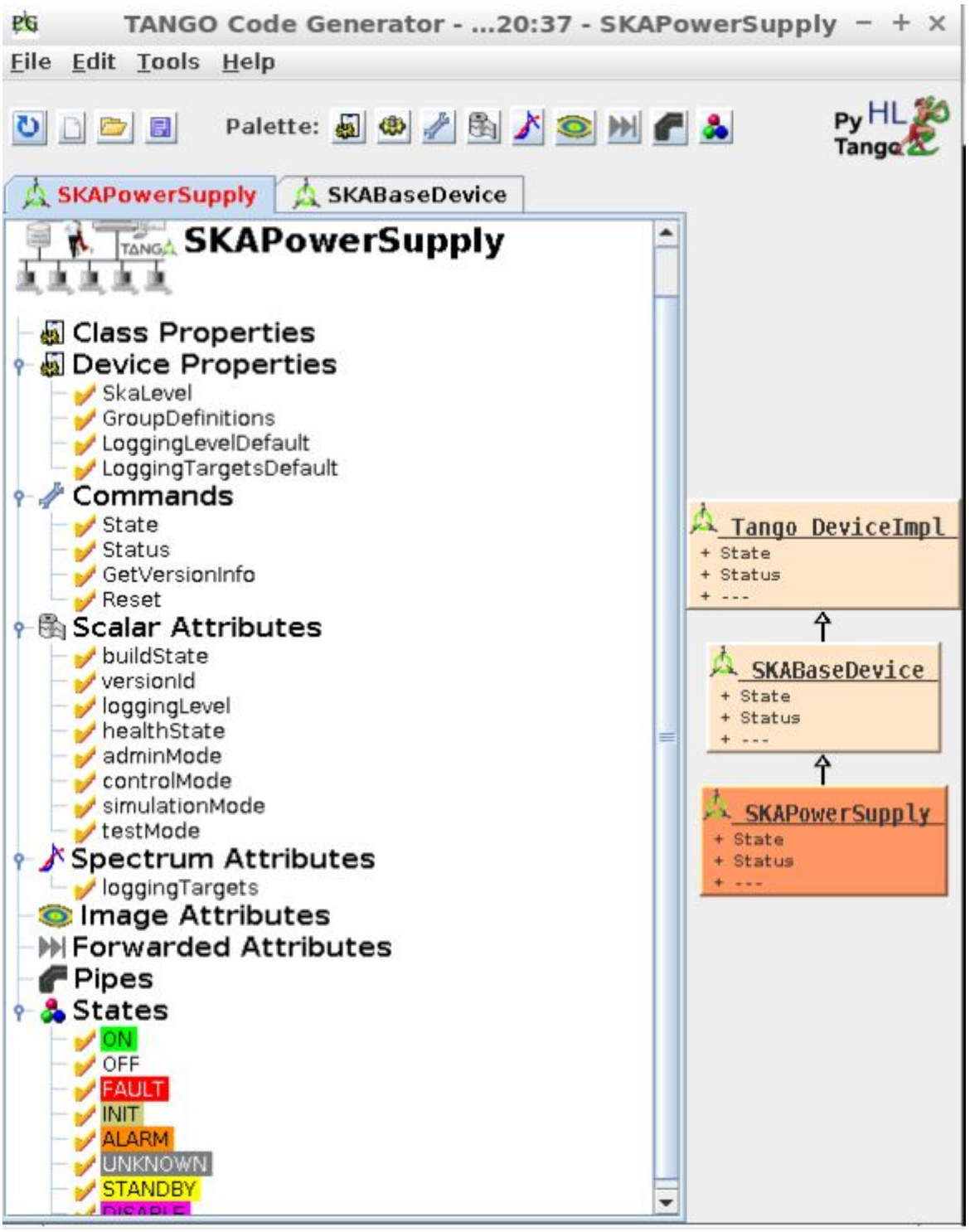
But, this mechanism will not be interpreted by Pogo, nor will be transparent to clients.

Other SKA Guidelines considerations

All SKA attributes must provide events to clients. The default method to achieve this will use the internal polling mechanism in Tango.

Exercise 7

Write a new PowerSupply Class, but inheriting from SKABaseDevice this time



Following the SKA Design Guidelines, all attributes should be polled. This will not have almost effect on pure-software attributes, so we will add this code just for fun:

```
   def read_attr_hardware(self,*args):
         print('read_hardware')
         time.sleep(2.)
```

Apply polling and events to all attributes, and try to ramp the power supply. What will happen?

Exercise 8: Move the ramp to a separate thread, pushing events in background.

Reimplement the write_Current method using a background thread:

```
   def write_Current(self, value):
        # PROTECTED REGION ID(FixedCurrentPowerSupply.Current_write) ENABLED START
#
        if self.get_state() == DevState.OFF:
            raise Exception("PS is OFF!")

        if self.RampingSteps > 0:
            self.ramp_thread = threading.Thread(
                target = self.ramp_current,
                args = (self.current, value),
                )
            self.ramp_thread.start()
        else:
            self.current = 0
            self.push_change_event('Current',self.current)
        # PROTECTED REGION END #    //  FixedCurrentPowerSupply.Current_write
```

See the effects on ipython/itango visualizing the event change.

Tango Groups

Groups can be easily created from ipython / iTango

```
In [15]: gr = PyTango.Group('test')

In [17]: gr.add('test/state/*')

In [18]: gr.get_device_list()
Out[18]: ['test/state/01', 'test/state/02', 'test/state/03', 'test/state/04',
'test/state/05']

In [20]: gr.read_attribute('Now')
Out[20]: [GroupAttrReply(), GroupAttrReply(), GroupAttrReply(), GroupAttrReply(),
GroupAttrReply()]

In [21]: rr = gr.read_attribute('Now')

In [22]: r = rr[0]

In [23]: r.get_data()
Out[23]: DeviceAttribute(data_format = tango._tango.AttrDataFormat.SCALAR, dim_x =
1, dim_y = 0, has_failed = False, is_empty = False, name = 'Now', nb_read = 1,
nb_written = 0, quality = tango._tango.AttrQuality.ATTR_VALID, r_dimension =
AttributeDimension(dim_x = 1, dim_y = 0), time = TimeVal(tv_nsec = 0, tv_sec =
1583225870, tv_usec = 439468), type = tango._tango.CmdArgType.DevDouble, value =
1583225870.4393737, w_dim_x = 0, w_dim_y = 0, w_dimension =
AttributeDimension(dim_x = 0, dim_y = 0), w_value = None)

In [24]: r.get_data().value
Out[24]: 1583225870.4393737

In [25]: [r.get_data().value for r in rr]
Out[25]:
[1583225870.4393737,
 1583225873.5306761,
 1583225873.5303254,
 1583225873.530499,
 1583225873.5309603]
```

Exercise 9

Create a device server running 5 SKAPowerSupply devices. Manage all 5 power supplies together as a group.

Device Hierarchies

Exercise 10

Create a new device server with 2 classes, PowerSupply and SKAPowerSupply. Declare a device for each one and Run it.

> The Server to launch 2 classes is as simple as:

```
# -*- coding: utf-8 -*-
#
# This file is part of the FixedCurrentPowerSupply project
#
#
#
# Distributed under the terms of the GPL license.
# See LICENSE.txt for more info.

""" FixedCurrentPowerSupply
A dummy power supply for testing
"""

# PyTango imports
import PyTango
from PyTango import DebugIt
from PyTango.server import run

from FixedCurrentPowerSupply import FixedCurrentPowerSupply
from SKAPowerSupply import SKAPowerSupply

# ----------
# Run server
# ----------


def main(args=None, **kwargs):
    # PROTECTED REGION ID(FixedCurrentPowerSupply.main) ENABLED START #
    return run((FixedCurrentPowerSupply,SKAPowerSupply), args=args, **kwargs)
    # PROTECTED REGION END #    //  FixedCurrentPowerSupply.main

if __name__ == '__main__':
    main()
```

Exercise 11

We are going to create a "SKAComposerDevice". Inheriting from SKABaseDevice, with a DeviceList property to specify a list of devices, and three spectrum attributes: Currents, Voltages and States, to show up the states of all the devices listed.

> For this device two implementations of the TangoGroup are going to be tested. First one based on read_asynch / read_reply commands.

Exercise 12

Re-implement the previous device but adding MaxVoltage and MaxCurrent attributes. This time, use event subscribing and callbacks to update the values of the attributes.

Dynamic creation of attributes

Live demo on creating Dynamic Attributes from Python HL (that's new even for me!)

Dynamic Commands work only on Python Classic API (and it's very experimental yet)

Fandango

Fandango: Functional programming 4 Tango

git clone https://github.com/tango-controls/fandango fandango.git cd fandango.git && python setup.py install

or

pip install fandango

Features:

- It provided some PythonHL client functionalities before PythonHL existed
- Online evaluation of formulas in Dynamic Attributes/Commands
- Python API to some functionalities only existing in Java (Jive/Astor)
- MySQL Access
- Allows to script device creation/configuration
- Some Qt3/4 helpers (obsolete)
- Advanced EventCallbacks (experimental)
- ... and fancy types (Caseless dicts/lists, Structs, Singletones, ...)

Python2, release 14.8, in production

Python3, release 15.0 (from 13), experimental

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

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)

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 PyTangoHL

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.

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

```
add_device(self, dev_info) -> None

        Add a device to the database. The device name, server and class
        are specified in the DbDevInfo structure

        Example :
            dev_info = DbDevInfo()
            dev_info.name = 'my/own/device'
            dev_info._class = 'MyDevice'
            dev_info.server = 'MyServer/test'
            db.add_device(dev_info)

    Parameters :
        - dev_info : (DbDevInfo) device information
```

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

```
fn.tango.add_new_device(server, klass, device)

This methods mimics Jive UI forms:

    server: ExecutableName/Instance
    klass:  DeviceClass
    device: domain/family/member
```

```
e.g.:
    fandango.tango.add_new_device(
      'MyServer/test','MyDevice','my/own/device')
```

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 fandango as fn

fn.tango.add_new_device('DynamicDS/1','DynamicDS','test/dyn/1')
astor = fn.Astor()
host = fn.linos.MyMachine().hostname
astor.start_servers('DynamicDS/1',host=host)
astor.set_server_level('DynamicDS/1',level=3,host=host)
```

---

methods from fandango can also be launched linux shell:

```
$: fandango add_new_device DynamicDS/1 DynamicDS test/dyn/1

$: fandango put_device_property test/dyn/1 DynamicAttributes "T=t%10"

$: tango_servers $HOSTNAME start DynamicDS/1
```

```
tango-cs@tangobox:~$ fandango find_devices "*hdb*es*"

dserver/hdb++es-srv/1

tango-cs@tangobox:~$ tango_servers start $(fandango find_devices "*hdb*es*")

start of ['dserver/hdb++es-srv/1'] at *
Loading dserver/hdb++es-srv/1 devices

Starting : ['hdb++es-srv/1']
------------------------------------------------------------------------------
/home/tango-cs/.local/bin/tango_servers start dserver/hdb++es-srv/1: Done

tango-cs@tangobox:~$
```

```
fandango.find_devices('bo01/vc/*')

fandango.find_attributes('sr12*/*plc*')
```

```
fandango.get_matching_device_properties('sr12/vc/eps-plc-01','dynamic*')
```

```
$ fandango -l find_devices "bl00/*"

bl00/ct/alarms
bl00/ct/ccdalarms-01
bl00/ct/eps-plc-01
bl00/ct/eps-plc-01-mbus
bl00/door/01
bl00/door/02
bl00/door/03
bl00/eh/ccg-fcv-01
bl00/eh/diset-01-ccd
bl00/eh/diset-01-iba
bl00/eh/diset-02-ccd
bl00/eh/diset-02-iba
bl00/eh/fcv-01
bl00/eh/ip-diset-01
bl00/eh/ip-diset-02
bl00/eh/ip-ip100-01
```

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

```
BL00_AIR_PRESSURE:
    BL00/CT/EPS-PLC-01/PAAS_EH01_01_PS1<=4 or
BL00/CT/EPS-PLC-01/PAAS_EH01_01_PS2<=4

BL09_STATES:
    any([s<0 or str(s) in ('UNKNOWN','FAULT') for s in
      FIND(BL00/VC/ALL/State)+FIND(BL00/VC/VGCT-0*/State)
      +FIND(BL00/VC/IPCT-0*/State)])

BL09_START:
    BL00/VC/Elotech-01/Temperature_Max > 20

BL00_PRESSURES:
    any([p>8e-07 for p in BL00/VC/ALL/CCGPressures[1:]])
```

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

Plenty of useful methods:

```
$ fandango --list

...
fandango.tango.methods.check_attribute
fandango.tango.methods.check_attribute_cached
fandango.tango.methods.check_attribute_events
fandango.tango.methods.check_device
fandango.tango.methods.check_device_cached
fandango.tango.methods.check_device_events
fandango.tango.methods.check_device_list
fandango.tango.methods.check_host
fandango.tango.methods.check_property_extensions
fandango.tango.methods.check_starter
fandango.tango.methods.delete_device
fandango.tango.methods.device_command
fandango.tango.methods.get_alias_dict
fandango.tango.methods.get_alias_for_device
fandango.tango.methods.get_attr_name
fandango.tango.methods.get_attribute_config
fandango.tango.methods.get_attribute_events
```

Links to Course Resources

Ubuntu Images

https://www.osboxes.org/lubuntu

UBUNTU:
https://sourceforge.net/projects/osboxes/files/v/vb/55-U-u/18.04/18.04.2/18042.64.7z/download

LUBUNTU:
https://sourceforge.net/projects/osboxes/files/v/vb/33-Lb--t/18.04/18.04.3/L1804.3-64bit.7z/download

PyTango Documentation

- https://pytango.readthedocs.io/en/stable/quicktour.html
- https://github.com/ajoubertza/icalepcs-workshop/blob/gh-pages/slideshow.md
- https://github.com/ajoubertza/icalepcs-workshop/tree/gh-pages/examples

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

Other Tango Tools and libraries

- https://github.com/tango-controls-hdbpp
- https://github.com/tango-controls/fandango
- https://github.com/tango-controls/panic

SKA Docker

- SKA developer portal: https://developer.skatelescope.org/en/latest/
- https://developer.skatelescope.org/en/latest/tools/tango-devenv-setup.html

SKA Base Classes

- SKA Python coding guidelines: https://developer.skatelescope.org/en/latest/development/python-codeguide.html
- https://gitlab.com/ska-telescope/lmc-base-classes
- https://gitlab.com/ska-telescope/lmc-base-classes/-/blob/master/skabase/SKABaseDevice/SKABaseDevice.py
- https://gitlab.com/ska-telescope/lmc-base-classes/-/tree/master/skabase/SKATestDevice

SKA Tango example

- https://gitlab.com/ska-telescope/tango-example
- https://gitlab.com/ska-telescope/tango-example/-/blob/master/README.md
- https://developer.skatelescope.org/projects/tango-example/en/latest/?badge=latest

Webjive

- https://gitlab.com/MaxIV/webjive
- https://github.com/ska-telescope/webjive
- https://webjive.readthedocs.io/en/latest/

WebJive tutorial

- https://developer.skatelescope.org/projects/ska-engineering-ui-compose-utils/en/latest/device.html