PyTango

www.tinyurl.com/PyTango

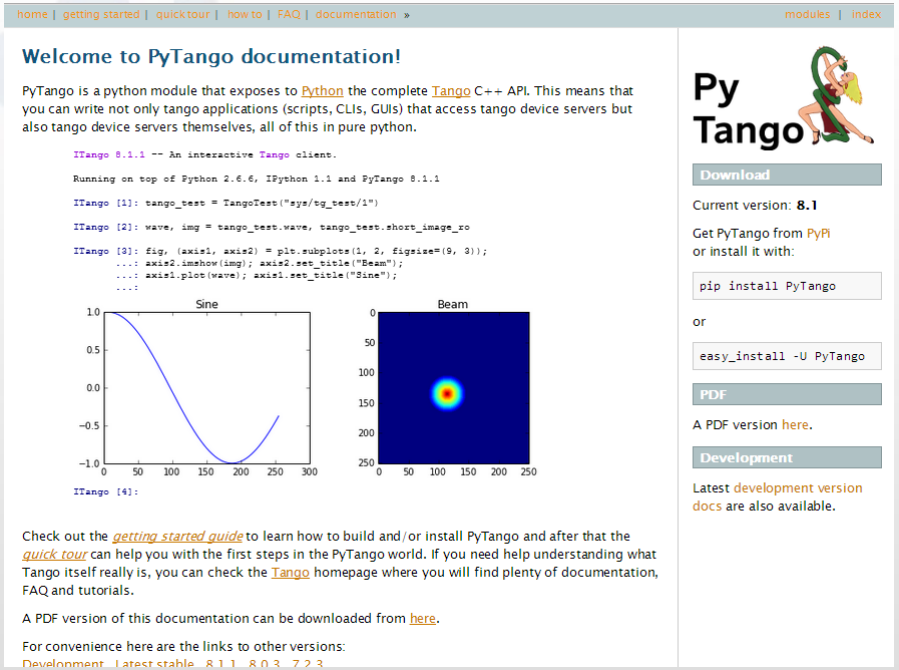
Agenda

- New in 8.1.2 (since 8.0.3)
- Server API
- DatabaseDS
- Cooperative API
- Feature requests
- Conclusions
- Questions

8.1.2

- Full Tango C++ 8.1 API
- 18 bug fixes
- 4 feature requests
- Server API
- Client green API

- Updated documentation
  - Quick tour **[revised]**
  - How to **[new]**
  - Server API **[revised]**
  - Client API **[revised]**



(missing windows binaries. Soon!)

Server API

Original PyTango Server API: Norman's door

```
import time
import PyTango
from PyTango import Device_4Impl
from PyTango import DeviceClass


class Clock(Device_4Impl):

  def __init__(self, d_class, name):
    Device_4Impl.__init__(self, d_class, name)
    self.init_device()

  def delete_device(self):
    pass

  def init_device(self):
    device_class = self.get_device_class()
    self.get_device_properties(device_class)

  def read_time(self, attr):
    attr.set_value(time.time())

  def strftime(self, format):
    return time.strftime(format)
```

```
class ClockClass(DeviceClass):

  attr_list = {
    'time':
      [[PyTango.DevDouble,
        PyTango.SCALAR,
        PyTango.READ]] }

  cmd_list = {
    'strftime':
      [[PyTango.DevString, ""],
       [PyTango.DevString, ""]] }

  def __init__(self, name):
    PyTango.DeviceClass.__init__(self, name)
    self.set_type(name)

if __name__ == '__main__':
  try:
    py = PyTango.Util(sys.argv)
    py.add_class(MotorClass, Motor, 'Motor')
    U = PyTango.Util.instance()
    U.server_init()
    U.server_run()

  except PyTango.DevFailed as df:
    print('DevFailed exception:')
    print(str(df))
  except Exception as e:
    print('Exception occurred:')
    print(str(e))
```

New server API (TEP 1)

Finished!

```
import time
from PyTango.server import run
from PyTango.server import Device, DeviceMeta
from PyTango.server import attribute, command


class Clock(Device):
    __metaclass__ = DeviceMeta

    @attribute
    def time(self):
        return time.time()

    @command(dtype_in=str, dtype_out=str)
    def strftime(self, format):
        return time.strftime(format)


if __name__ == "__main__":
    run([Clock])
```

attributes

- python `property` *like* API
- Default: scalar, double, read-only:

```
class Clock(Device):

  @attribute
  def time(self):
    return time.time()
```

```
class Clock(Device):

  time = attribute()

  def read_time(self):
    return time.time()
```

- Fully configurable:

```
class Clock(Device):

  @attribute(label="Time", dtype=float,
             unit="s")
  def time(self):
    """time since epoch"""
    return time.time()

  @time.write
  def time(self, new_time):
    pass
```

```
class Clock(Device):

  time = attribute(label="Time", dtype=float,
                   doc="time since epoch", unit="s",
                   access=AttrWriteType.READ_WRITE)

  def read_time(self):
    return time.time()

  def write_time(self, new_time):
    pass
```

commands

```
class Clock(Device):

  @command
  def resetTime(self):
    pass

  @command(dtype_in=str,  doc_in="time format",
           dtype_out=str, doc_out="formatted current time")
  def strftime(self, format):
    return time.strftime(format)
```

properties

```
class PowerSupply(Device):

  host = device_property(dtype=str)

  port = class_property(dtype=int, default_value=9788)

  def init_device(self):
    Device.init_device(self)
    self.info_stream("Connect to %s:%d", self.host, self.port)
```

server loop

- Single line server loop:

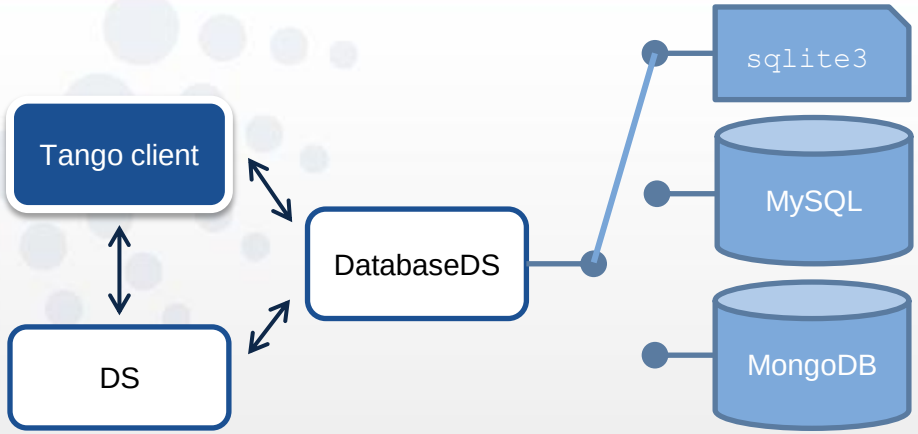
```
if __name__ == "__main__":
    run([Clock])
```

- Fully configurable:

```
# -- MyDS.py --

from ClockDS import Clock
from PowerSupplyDS import PowerSupply
from OldDS import OldDevice, OldDeviceClass

def event_loop_cb(): pass

def post_init_cb():  pass

if __name__ == "__main__":
    devices = {"Clock": Clock,
               "PowerSupply": PowerSupply,
               "AncientDevice": (OldDevice, OldDeviceClass)}

    util = run(devices, args=sys.argv, verbose=False,
               event_loop=event_loop_cb,
               post_init=post_init_cb,
               util=None)
```

DatabaseDS (TEP2)

- Python Database DS
  - Standalone system
  - Small/medium installations
  - Demonstrations
  - Prototyping
  - Integration with other systems
- 99% API like the C++ counterpart
  - Less scalable, less performance
  - More flexible
- Status
  - It runs!
  - Databaseds 100%
  - sqlite3 backend 75% (needs testing)



Maria-Teresa ★★★★★

S. Petitdemange ★★★★★

```
$ DataBaseds 2 -ORBendPoint giop:tcp::10000 --db=[sqlite3, mongodb, ...]
```

Cooperative API (TEP 5)

- Green API

- Two *green* modes
  - Futures (`concurrent.futures`)
  - Gevent (`gevent` >= 1.0)

- Only client side: DeviceProxy
  - `Constructor, ping, state, status`
  - `read_attribute(s), write_attribute(s)`
  - `write_read_attribute`
  - `(un)subscribe_event`

M. Guijarro ★★★★★

concurrent.futures (PEP 3148)

a high-level interface for asynchronously executing callables

- Submit a *callable* and get back a **Future** object
- New in python 3.2
- Backport on PyPi

```
>>> from PyTango.futures import DeviceProxy

>>> dev = DeviceProxy("sys/tg_test/1")
>>> result = dev.state(wait=False)
>>> result
<Future at 0x16cb310 state=pending>

>>> # this will be the blocking code
>>> state = result.result()
>>> print(state)
RUNNING

>>> result = dev.read_attribute('wave', wait=False)
>>> result
<Future at 0x16cbe50 state=pending>

>>> # blocking here again
>>> dev_attr = result.result()
>>> print(dev_attr)
DeviceAttribute(....)
```

gevent

a coroutine-based python networking library

- Spawn a task and get back an AsyncResult
- Uses *greenlets (micro-threads, tasklets)*
- Based on *libev* (epoll/kqueue)
- Cooperative sockets and ssl (monkey_patch)

```
>>> from PyTango.gevent import DeviceProxy

>>> dev = DeviceProxy("sys/tg_test/1")
>>> # Amazing: this I/O will be NON blocking
>>> state = dev.state()
>>> print(state)
RUNNING

>>> # Amazing: this I/O will be NON blocking
>>> wave = dev.read_attribute('wave')
>>> print(wave)
DeviceAttribute(....)
```

Feature requests

- Server
  - Know which client is making a request
  - Know if a client (dis)connects
  - Plug a file descriptor in the CORBA mainloop (select/epool)
  - read attribute / command have a callback to notify data can be released
  - Fully dynamic devices
    - `Util.add_class(any_class)`
    - `Util.add_device(any_object)`
    - at any time in server life cycle
  - Run in single threaded mode
  - Pogo code generation with setup.py
  - Pogo code generation for new server API

Feature requests

- Client
  - Read a slice of an attribute (1D/2D)
    - `roi = device.spectrum_attr[start:end]`
    - `roi = device.image_attr[startCol:endCol][startRow:endRow]`
- Miscellaneous
  - Host resolution (DNS request) in the client (corbaloc instead of IOR)
  - auto-discovery of TANGO_HOST if not defined (on local network)
  - structured attribute and command

Conclusion: Turing Machine server

```python
import json
from PyTango import DevState, AttrWriteType
from PyTango.server import Device, DeviceMeta, run
from PyTango.server import attribute, command
from PyTango.server import device_property

class TuringMachine(Device):
  __metaclass__ = DeviceMeta

  final_states = attribute(dtype=[str],
                   access=AttrWriteType.READ_WRITE)

  def init_device(self):
    Device.init_device(self)
    self.__tape = None
    self.__head = -1
    self.__state = "init"
    self.__final_states = None
    self.__transition_function = None

  def read_final_states(self):
    return self.__final_states

  def write_final_states(self, final_states):
    self.__final_states = final_states

  @attribute(dtype=str)
  def transition_function(self):
    return self.__transition_function

  @transition_function.write
  def transition_function(self, func_str):
    self.__transition_function = tf = {}
    for k, v in json.loads(func_str).items():
      tf[tuple(str(k).split(","))] = map(str, v)

  @attribute(dtype=str)
  def tape(self):
    s, keys = "", self.__tape.keys()
    for i in range(min(keys), max(keys)):
      s += self.__tape.get(i, " ")
    return s

  @tape.write
  def tape(self, new_tape):
    self.__head = 0
    self.__state = "init"
    self.__tape = tape = {}
    for i in range(len(new_tape)):
      tape[i] = new_tape[i]

  @command
  def step(self):
    head_char = self.__tape.get(self.__head, " ")
    y = self.__transition_function.get(
                        (self.__state, head_char))
    if y:
      self.__tape[self.__head] = y[1]
      if y[2] == "R":
        self.__head += 1
      elif y[2] == "L":
        self.__head -= 1
      self.__state = y[0]

  def dev_state(self):
    if self.__state == "init":
      return DevState.INIT
    elif self.__state in self.__final_states:
      return DevState.ON
    return DevState.RUNNING

run([TuringMachine])
```

Conclusion: Turing Machine client

```
import json
from PyTango import DevState
from PyTango.gevent import DeviceProxy

turing = DeviceProxy("demo/turing/1")

tf = {"init,0":("init", "1", "R"),
      "init,1":("init", "0", "R"),
      "init, ":("final"," ", "N"),}

turing.transition_function = json.dumps(tf)
turing.tape = "010011 "
turing.final_states = ['final']

print("Input on Tape:")
print(turing.tape)

while not turing.state() == DevState.ON:
    turing.step()

print("Turing machine calculated:")
print(turing.tape)
```

Questions