

PyTango



Tiago Coutinho
Ramon Suñé

- Recent changes
- Future changes

- ❏ Updated TangoC++ APIs
- ❏ API improvements
- ❏ Numpy
- ❏ DeviceProxy: Events
- ❏ Dynamic Attributes
- ❏ DeviceImpl: Logger support
- ❏ Documentation
- ❏ Distribution

API Improvements:

- ■ `read_attribute` returns `DeviceAttribute` with:

- 'value' and 'w_value' instead of concatenated 'value' and 'w_scalar_value'
- 'time' has type `TimeVal`, which offers handy methods:
 - ■ `.totime()`, `.todatetime()`, `.fromtimestamp()`, `.fromdatetime()`...

- ■ `read_attributes` does not throw exceptions. `read_attribute` does.

API Improvements:

DeviceImpl.get_write_value() improved API

```
class MyImpl(PyTango.Device_4Impl):  
  
    def write_Attribute(self, attr):  
  
        # Old Style:  
        data = []  
        self.get_write_attribute(data)  
        print 'value written:', data[0]  
  
        # New style  
        value = self.get_write_attribute()  
        print 'value written:', value
```

API Improvements:

- `r = d_proxy.commandName(param)`
- `v = d_proxy['Attribute'].value`
- `d_proxy['Attribute'] = 75`
- Type info: `is_int_type`, `is_scalar_type...`

- Updated TangoC++ APIs. Exposes:
 - APIs new in Tango 7
 - `dev.write_read_attribute(...)`
 - `dev.lock()`, `dev.unlock()`
 - `DATA_READY_EVENT`
 - APIs not fully supported in older versions
 - `PyTango.Group`
 - `PyTango.Database`

❏ Numpy

- ❏ For attributes

- ❏ For commands

```
DeviceProxy.read_attribute(  
    self, attr_name, extract_as=ExtractAs.Numpy)  
  
a = d.read_attribute('SmallArray', extract_as=ExtractAs.List)  
  
DeviceProxy.command_inout(cmd_name, cmd_param=None)  
  
# param can be anything. If the command accepts a  
# DevVarLongArray, it can be a numpy of int32(best)  
# or a list of ints or a DeviceData with a numpy  
# of int32(best) or list of ints has been inserted  
r = d.command_inout_raw('cmd_name', param)  
    .extract(ExtractAs.List)
```

■ Numpy

- For attributes

- For commands

```
NumpyType.DevULong  
NumpyType.DevDouble
```

```
...
```

```
a = numpy.array([1,2,3], dtype=PyTango.NumpyType.DevDouble)
```

```
NumpyType.tango_to_numpy(almost_anything)
```

```
NumpyType.spectrum(tg_type, dim_x)
```

```
NumpyType.image(tg_type, dim_x, dim_y)
```

```
a = PyTango.NumpyType.spectrum(PyTango.DevLong, 10)
```

DeviceProxy: Events

```
DeviceProxy.subscribe_event(  
    self,  
    attr_name,  
    event,  
    callback,  
    filters=[],  
    stateless=False,  
    extract_as=ExtractAs.Numpy)  
  
def callback(event):  
    print 'new event!'  
  
d = DeviceProxy('A/TG/DEV')  
d.subscribe_event('State', callback, EventType.CHANGE_EVENT)  
  
d = None # Automatic unsubscription
```

❏ Dynamic Attributes

- ❏ Easy definition.
- ❏ Methods now searched when needed.
- ❏ Also access allowed groups.

Recent Changes

12

```
def _is_AT_allowed(self, attr, req_type):
    return self.get_state() not in [PyTango.DevState.UNKNOWN]

def _read_dynamic(self, attr, id):
    self.info_stream('reading', id)
    attr.set_value(id)

def _write_dynamic(self, attr, id):
    value = attr.get_write_value()
    self.info_stream('writing', id, '. Value is', value)

def _add_dynamic_attributes(self):
    self.add_attribute(
        'Attr0',
        r_meth=lambda a: self._read_dynamic(a, 0),
        w_meth=lambda a: self._write_dynamic(a, 0),
        is_allo_meth=self._is_AT_allowed )
    self.add_attribute(
        'Attr1',
        r_meth=lambda a: self._read_dynamic(a, 1),
        w_meth=lambda a: self._write_dynamic(a, 1),
        is_allo_meth=self._is_AT_allowed )

def __init__(self):
    self._add_dynamic_attributes()
```

Dynamic Attributes

```
def _is_default_allowed(self, attr, req_type):
    if self.get_state() in [
        PyTango.DevState.UNKNOWN,
        PyTango.DevState.FAULT]:
        return False
    return True

def _set_default_state_machine(self):
    klass = self.get_device_class()
    for att_name in klass.attr_list:
        fn_name = 'is_' + att_name + '_allowed'
        if not hasattr(self, fn_name):
            setattr(self,
                    fn_name,
                    self._is_default_allowed)

def __init__(self):
    self._add_dynamic_attributes()
    self._set_default_state_machine()
```

☒ Useful decorator

```
def tango_trace(fn):
    def __doit(self, *args, **kwds):
        fname = self.get_name() + "::" + fn.__name__ + "()"
        self.debug_stream("Entering ", fname, "...")
        try:
            fn(self, *args, **kwds)
        except Exception:
            self.debug_stream("...Leaving", fname, "[EXCEPTION]")
            self.debug_stream(traceback.format_exc())
            raise
        self.debug_stream("...Leaving", fname)
    return __doit

@tango_trace
def write_attribute(self, attr): pass
```

DeviceImpl: Logger support

```
DeviceImpl.debug_stream(self, *args)
```

```
x = 3
```

```
# Past
```

```
print 'old style debug message'
```

```
# Present
```

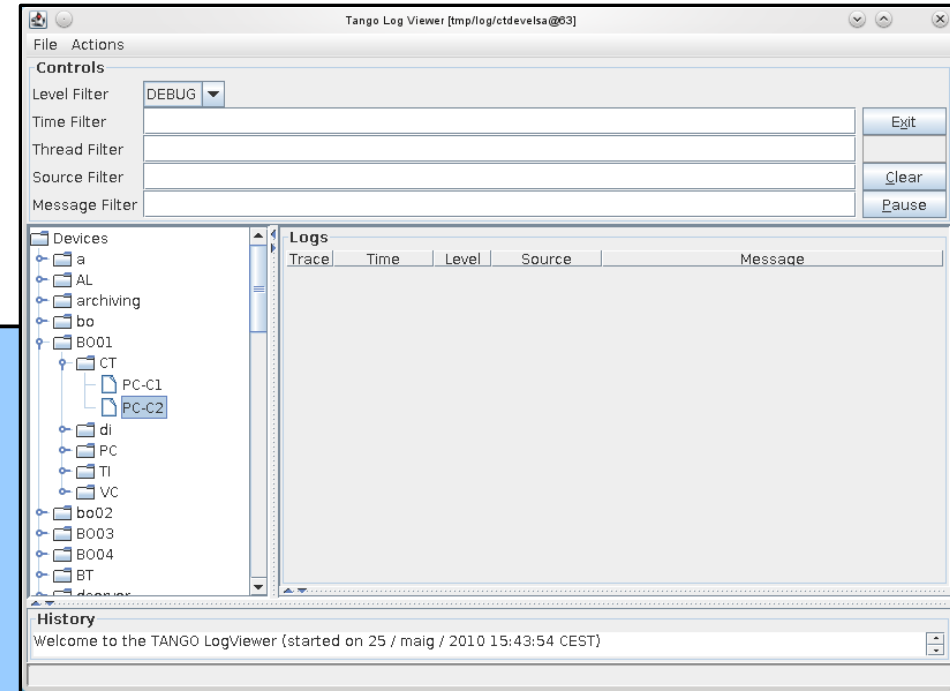
```
self.debug_stream('debug message. X is now', x)
```

```
self.info_stream('info message', 'more strings', 'more')
```

```
self.warn_stream('warn message')
```

```
self.error_stream('error message')
```

```
self.fatal_stream('fatal message')
```



❏ Documentation

- Based on Sphinx
- Tutorial-like + API doc
- API doc also in help()
- Nice looking in help() and html
- Has a search feature

Welcome to PyTango 7.1 documentation!



Indices and tables

- [Index](#)
- [Module Index](#)
- [Search Page](#)

PyTango

The python Tango binding is made accessible using the module PyTango.

The PyTango python module implements the Python Tango Device and Database API mapping but also allow to write Tango device server using Python.

It then allows access from Python environment to the Tango high level C++ classes and structures (see "The TANGO Control system manual" for complete reference).

If you want to write Tango device server in Python using this PyTango module, you need Tango C++ library release 7.1 or above.

If you simply need to write Python script which act as Tango client only, older releases of tango can be used.

These Tango high level C++ classes and structures are exported to Python using the Boost.python library (see <http://boost.sourceforge.net>). Details on how to install the boost library and generate the python PyTango module from source code are available in the Readme files of this packages.



Table Of Contents

- Welcome to PyTango 7.1 documentation!
- [Indices and tables](#)
 - [PyTango](#)
 - [Revision](#)

Next topic

[Getting started](#)

This Page

[Show Source](#)

Quick search

Go

Enter search terms or a module, class or function name.

As you can see in the following example, when scalar types are used, the Tango binding automatically manages the data types, and writing scripts is quite easy:

```

1 from PyTango import *
2 import sys, os, time
3
4 tangotest = DeviceProxy("tango/tangotest/1")
5
6 # First use the classical command_inout way to execute the DevString command
7 # (DevString in this case is a command of the TangoTest device)
8
9 result= tangotest.command_inout("DevString", "First hello to device")
10 print "Result of execution of DevString command=", result
11
12 # the same with a Device specific command
13 result= tangotest.DevString("Second Hello to device")
14 print "Result of execution of DevString command=", result
15
16 # Please note that argin argument type is automagically managed by python
17 result= tangotest.DevULong(12456)
18 print "Result of execution of Status command=", result

```

Execute commands with more complex types

In this case you have to use put your arguments data in the correct python structures:

```

1 from PyTango import *
2 import sys, os, time
3
4 print "Getting DeviceProxy "
5 tango_test = DeviceProxy("tango/tangotest/1")
6 # The input argument is a DevVarLongStringArray
7 # so create the argin variable containing
8 # an array of longs and an array of strings
9 argin = ([1,2,3], ["Hello", "TangoTest device"])
10
11 result= tango_test.DevVarLongArray(argin)
12 print "Result of execution of DevVarLongArray command=", result

```

Reading and writing attributes

Basic read/write attribute operations:

```

1 #Read a scalar attribute

```

Search

From here you can search these documents. Enter your search words into the box below and click "search". Note that the search function will automatically search for all of the words. Pages containing fewer words won't appear in the result list.

Search Results

Search finished, found 19 page(s) matching the search query.

- [PyTango.DeviceProxy.read_attribute](#) (method, in DeviceProxy)
- [PyTango.DeviceProxy.read_attribute_async](#) (method, in DeviceProxy)
- [PyTango.DeviceProxy.read_attribute_reply](#) (method, in DeviceProxy)
- [PyTango.DeviceProxy.read_attributes](#) (method, in DeviceProxy)
- [PyTango.DeviceProxy.read_attributes_async](#) (method, in DeviceProxy)
- [PyTango.DeviceProxy.read_attributes_reply](#) (method, in DeviceProxy)
- [PyTango.DeviceProxy.write_read_attribute](#) (method, in DeviceProxy)
- [PyTango.Group.read_attribute](#) (method, in Group)
- [PyTango.Group.read_attribute_async](#) (method, in Group)
- [PyTango.Group.read_attribute_reply](#) (method, in Group)
- [PyTango.Group.read_attributes](#) (method, in Group)
- [PyTango.Group.read_attributes_async](#) (method, in Group)
- [PyTango.Group.read_attributes_reply](#) (method, in Group)
- [AttributeProxy](#)
AttributeProxy ----- .. currentmodule:: PyTango .. autoclass:: PyTango.AttributeProxy
:members: ...
- [DeviceProxy](#)



DeviceProxy — PyTango v7.1.0 documentation - Mozilla Firefox

Fitxer Edita Visualitza Historial Adreces d'interès Eines Ajuda

http://www.tango-controls.org/static/PyTango/latest/doc/html/client/device_proxy.html?highlight=read_attribute#PyTango.DeviceProxy.read_attribute

Més visitades timing Tango Client TODO /references/ Tango Device Server ... RT at a glance Note in Reader [Controls] Index of fr... Welcome to PyTango ... MIS Applications — C...

DeviceProxy — PyTango v7.1.0 documenta...

```

read_attribute(value, extract_as=PyTango._PyTango.ExtractAs.Numpy)
read_attribute (self, attr_name, extract_as=ExtractAs.Numpy) -> DeviceAttribute

    Read a single attribute.

    Parameters:attr_name: (str) The name of the attribute to read.
                extract_as: (ExtractAs) Defaults to numpy.

    Return:     (DeviceAttribute)

    Throws:    ConnectionFailed, CommunicationFailed, DevFailed from device

read_attribute_asynch(attr_name, cb=None)
read_attribute_asynch ( self, attr_name) -> int
read_attribute_asynch ( self, attr_name, callback) -> None

    Shortcut to self.read_attributes_asynch([attr_name], cb)
  
```

ctdevelsa (rsune) {Ctrl+F11}

Fitxer Edita Visualitza Història Punts Arranjament Ajuda

```

Help on method __DeviceProxy__read_attribute in module PyTango.device_proxy:

__DeviceProxy__read_attribute(self, value, extract_as=PyTango._PyTango.ExtractAs.Numpy) method of PyTango._PyTango.DeviceProxy instance
  read_attribute(self, attr_name, extract_as=ExtractAs.Numpy) -> DeviceAttribute

    Read a single attribute.

    Parameters :
      - attr_name : (str) The name of the attribute to read.
      - extract_as : (ExtractAs) Defaults to numpy.
    Return      : (DeviceAttribute)

    Throws     : ConnectionFailed, CommunicationFailed, DevFailed from device
  
```

lines 1-13/13 (END)

PyTango-trunk : vi ...enta-PyTango2 : bash ctdevelsa (rsune)

❏ Distribution:

❏ setup.py

```
$ python setup.py build  
$ python setup.py install
```

❏ setuptools

```
$ python easy_install -U PyTango
```

❏ Debian Package

- Thanks to Frédéric-Emmanuel Picca

- ❏ Missing Tango C++ features
- ❏ Automatic test procedure
- ❏ Python ≥ 3.0 support
- ❏ New Pogo
- ❏ Alternatives

❏ Missing TangoC++ features

- ❏ DevEncoded JPEG support

- ❏ Mix C++ and Python in the same server

- ❏ Dynamic devices:

- Requested by *S. Petitdemange*

- You can add a new device to the Database

- But you can't do `DevClass.device_factory('dev/na/me')`

- ❏ Anything not in TangoC++-7.1

- ❏ Automatic test procedure
 - ❏ Not even decided the unit test framework to use
 - ❏ For now we have an old script that should be updated

Python \geq 3.0 support

Why?

- It's the future!

Can we do it?

- ✓ python
- ✓ boost-python
- ✗ ~~numpy~~
- ✓ PyQt4

■ New Pogo

■ No news yet!



❏ Alternative implementation

- Currently using boost-python.
- Big call stack, hard to navigate backtrace when debugging.
- Very flexible. Not so fast.
- Windows issues:
 - No windows binaries
 - Latest versions DLL hell with IE7

❏ Alternative implementation

■ CPython:

- Would 'potentially' generate the most efficient code
- Need an army of programmers (knowledgable of Python C API)

■ SWIG:

- Easy to do small simple things
- Easy to integrate in different build systems
- Specification files are 'C++ like' files.
- Need to learn some tricks to get around STL classes and to manage namespaces

❏ Alternative implementation

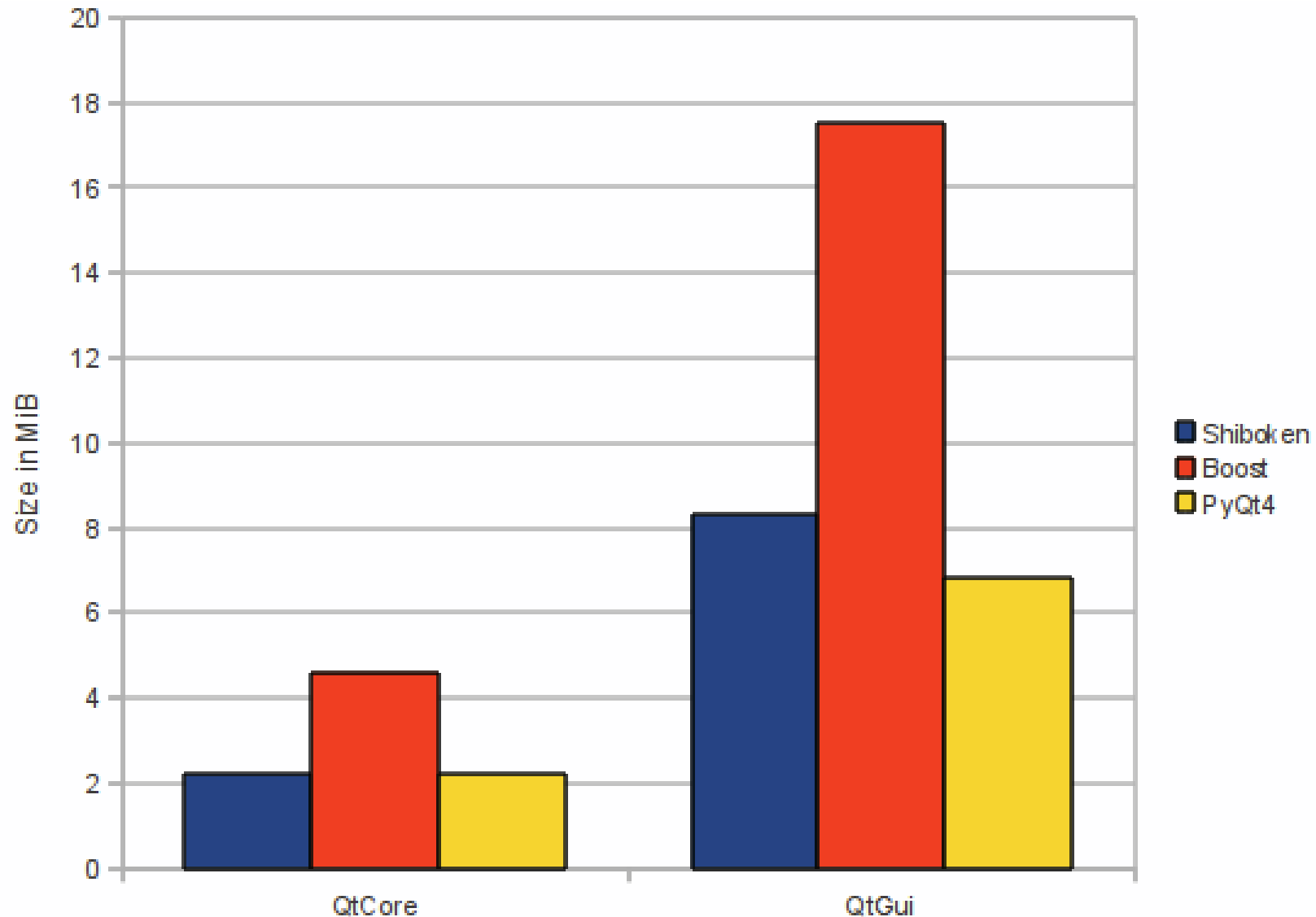
■ SIP:

- Specification files are 'C++ like' files.
- Very powerful
- Proven in big projects (PyQt)
- Generates the most efficient code
- Active development

■ Shiboken:

- New kid on the block
- CPython based
- Not yet in production phase
- Looks promising

Alternative implementation



❏ Alternative implementation

❏ cPyTango

- What it is NOT:
 - ❏ A replacement binding for PyTango
 - ❏ 'Finished' (in fact, it's not even started)
 - ❏ 'Stable'/'Mature'/'Ready for production'
 - ❏ Documented
 - ❏ There were no performance tests
- What it is:
 - ❏ Based on Jens minimal client ctango binding
 - ❏ 3 python files (800 lines of undocumented code)
 - ❏ Uses python native ctypes
 - ❏ Needs ctango binding library
 - ❏ ... but, it is 'ready to playing with'

