Tango training

client API - python

Elettra Sincrotrone Trieste

Claudio Scafuri

claudio.scafuri@elettra.eu

Elettra
Sincrotrone
Trieste

outline

remarks

fundamental operations with examples

Elettra
Sincrotrone
Trieste

Python Binding

Based on the C++ api and boost library for C++ to python link

- core is C++ Tango library (reference impl.)
- complete coverage of features (almost?)
  - both for clients and servers
- integrated with numpy library
  - Spectrum and Image are numpy.ndarray
- interactive python shell
  - itango : lots of goodies
  - great for testing, debugging, system maintenance

Elettra
Sincrotrone
Trieste

python client side API

Each Tango device is an instance of ***PyTango.DeviceProxy***

- Easy connection building between clients and devices
- Manages re-connection

```
dev = PyTango.DeviceProxy('sys/tg_test/1')
```

Elettra
Sincrotrone
Trieste

Errors

Errors throw exceptions (clean, standard O.O. style)

Exceptions derived from ***PyTango.DevFailed*** class:

- one *excpet* is enough for Tango errors
- 10 exceptions classes derived from DevFailed allow detailed analysis:

ConnectionFailed, CommunicationFailed, WrongNameSyntax,

NonDbDevice, WrongData, NonSupportedFeature,

AsynCall, AsynReplyNotArrived, EventSystemFailed,

NamedDevFailedList

see documentation

Elettra Sincrotrone Trieste

reading an attribute

- *DeviceProxy read_attribute* method
- class DeviceAttribute is used for the received data
- argument is the name of the attribute

```
da = dev.read_attribute('double_scalar')
val = da.value
```

- `da` contains lots of information (same as C++)

- simplified syntax for reading directly the value:

```
val = dev.double_scalar
```

  - but you loose all the additional information

see: reading01.py

Elettra Sincrotrone Trieste

reading an attribute

- DeviceAttribute contains **lots** of informations:
  - name of attribute
  - timestamp
  - dim_x, dim_y
  - type
  - quality factor
  - setpoint of a r_w variable IMPORTANT !!
- PyTango handles some types in way that is "more friendly" than C++ (e.g. time...)

see example: reading02.py

reading vector (spectrum)

- very easy to handle using numpy.ndarray!
  - memory management done by python
  - no need to declare special types of data
  - use a simple assignment:

```
value = dev.long64_spectrum_ro
```

see example: reading03.py

Elettra Sincrotrone Trieste

writing an attribute

with DeviceProxy write*_attribute* method

```
value = 0.618
attname = 'double_scalar'
void DeviceProxy.write_attribute(attname,value)
```

You can also use the symplified Pythonic syntax :

```
dev.double_scalar=0.618
```

Simpler and more intuitive then in C++

but do a double check on the attribute name!

see example: writing01.py

Elettra Sincrotrone Trieste

writing vector (spectrum)

data is passed to the call by means of a ***python list*** or a ***numpy.ndarray*** object

They must contain elements of the correct or at least compatible type

see example: writing02.py

Elettra
Sincrotrone
Trieste

executing commands

DeviceProxy command_inout method sends a command to the device. It may return a value, depending on server implementation:

```
DeviceProxy.command_inout(cmdname,value)
```

Pay attention to types!

see example: command01.py, command02.py

Elettra
Sincrotrone
Trieste

asynch execution

In python too the asynch version of read_attribute, write_attribue and command_inout are available.

Data retrival from asynch calls follows the same models and concpts of the C++ api: polling, callback pull and callback push.

see example: asyncmd01.py

Elettra
Sincrotrone
Trieste

event subscription

Clients do not poll: polling is done in Server : polling thread.

Server must be configured/written to generate events.

Server sends data to interested client when “something” changes.

Clients must *subscribe* to events and handle *asynchronous* data.

Elettra
Sincrotrone
Trieste

event subscription

Client creates an object (a callable) implementig the `push_event()` method, which is executed upon event arrival.

Then client calls `subscribe_event` on the DeviceProxy

Upon successful subscrition at least one event is sent by server.

Client calls `unsubscribe_event()` when no more events are needed.

Elettra Sincrotrone Trieste

event subscription

By default Event subscrition fails if Server does not support the requested event.

- if *stateless* is set to True subscrition succeeds anyway, and events will be recived when the server is up and correctly configured!

Heartbeat : client is notified if server is not reachable via event (exception is notified to callback)

Hertbeat automatically re-subcribes event again when server becomes reachable again.

see: event01.py

Elettra
Sincrotrone
Trieste

Group

Provides a single point of control for a group of devices: client performs a ***read/write_attribute*** or ***command_inout*** to all the devices with a single call.

Group calls are internally executed asynchronously: speed!

Client creates a Tango::Group, then adds devices

It is also possible to add a Group object to a Group to create a hierarchical object (advanced feature).

Elettra
Sincrotrone
Trieste

Group

Devices are added by *name*.

Wildcard * can be used in any part of the name (*/*/* : whole control system).

Groups are really useful when:

- Devices have uniform interfaces: use AbstractClasses
- there is a consistent and rational naming convention for device names
    - chek Tango wildcard expansion rules, ordering.

-----> Design the control system as a <u>whole:</u> bottom-up AND top-down

Deployment and configuration of Devices is an important and delicate step in the building and maintenace of the control system.

Elettra
Sincrotrone
Trieste

Group

no exceptions during Group calls

error are reported by GroupRelplyList objects:

- check has_failed() globally
- check has_failed() for each list elemen
  - can enable excpetions and get an exception while exctrating a datum:
  - `GroupReply::enable_exceptions(true);`

Elettra
Sincrotrone
Trieste

Group

Group can contain also other Groups

- hierarchical organization

The reply is not heriarchical!

Actions performed on a Group can be forwarded or not to subgroups

- default: forward
- turn off/on globally with statyc method on Group
- additional parameter for actions

Elettra
Sincrotrone
Trieste

Group

with a Group a client can:

- excute command
  - without argument
  - with same iput argument to all members
  - with different input arguments for different members
- read one attribute
- write one attribute
  - with same value for all members
  - with different values for different members

see: group01.cpp

comprehensive examples on Tango manual