Tango kernel status

- News from kernel
  - Tango 8
    - Attribute properties (Tomasz)
    - Event system
    - Miscellaneous
    - Pogo
Tango 8

• In use at ESRF
  • At least for machine control system
  • Don't be jealous, it's Tango 8.0.4 !!
  • Release available for the community: 8.0.5
List of changes

Tango 8 has been developed and tested using:

- omniORB 4.1.6
- zmq 3.1
- log4Tango 4.0.5

Changes between Log4Tango 4.0.3 and Log4Tango 4.0.5

- SourceForge bug 3156197
- Fix warnings when Tango is compiled -Wall -Wextra
- Add Windows port for Windows 64 bits VC10

Changes in Tango itself

- New event system based on ZMQ
- New methods to manage polling in DeviceImpl class (is_attribute_polled(), is_command_polled(), get_attribute_poll_period(), get_command_poll_period(), poll_attribute(), poll_command(), stop_poll_attribute(), stop_poll_command())
- DevEncoded data type supported for commands
- New Attribute class setter/getter methods for min_alarm, max_alarm, min_warning and max_warning attribute properties
- New Attribute set_properties/get_properties to set/get several attribute properties in one call
- Cleaner way to reset kernel attribute properties to lib/user/class default value
- Add some C++11 features when compiler support them (Lambda functions - unique_ptr for extension classes
- Move constructor and assignment for DeviceData and DeviceAttribute classes

This requires a new compilation option (-std=c++0x)

- Add device log messages when any device attribute(s) quality factor changes
  (ATTR_INVALID -> error stream, ATTR_CHANGING -> info stream, ATTR_VALID -> info stream
  ATTR_ALARM : min/max alarm -> error stream, min/max warning + rds -> warning stream)
- Add a clean_db parameter to the DeviceImpl::remove_attribute() method. Default is true
- New DeviceProxy::get_access_right() method
- New Util::is_svr_starting(), Util::is_svr_shutting_down() and Util::is_device_restarting() methods
- New DeviceClass::get_cmd_by_name() method
- New DServer::_create_cpp_class() method (For PyTango)
- Remove warnings compilation (Tango is now compiled with -Wall and -Wextra)
- Add Group::command_inout(), Group::command_inout_asynch(), Group::write_attribute() and
  Group::write_attribute_asynch() with vector<DeviceData> to carry the data.
- Improvements in event management for notifid events (link to bug 3293671)
- For writable and memorized attribute(s), check coherency of new min/max_value with memorized value when
  the attribute configuration is modified.
- State computation for device with alarmed attributes: If the attribute is polled, the attribute value is
  read from the polling buffer (also true when reading the state as a CORBA attribute)
- Add pre-processor define for Tango release number management (TANGO_VERSION_MAJOR, TANGO_VERSION_MINOR
  and TANGO_VERSION_PATCH)
- Host IP address(es) is(are) now retrieved from network interface(s)
- Add a check during set_attribute_config() call for users trying to change hard coded properties
- Optimization in DeviceProxy methods to get asynchronous call replies when caller uses a timeout in case the
  reply is already there
- Remove some “cerr” messages in AttributeProxy class
- Util::get_host_name() always returns host name in lower case letters
- The caller PID is now reported in black-box also when UNIX socket is used as transport
- write_attribute() called during device server startup sequence due to memorized attribute(s) is reported
  in black box with a specific message
- It’s now possible to poll command/attribute in a device server started without database for
  command/attributes
  with polling defined in code
- Add a polling thread tuning after the execution of UpdObjPollingPeriod command
- Remove all Solaris specific code
- Remove all old stream specific code
- Signals SIGUSR1 and SIGUSR2 can now be used within a device server process
- Optimize database calls during device server startup and shutdown sequence (When TAC is used or when
  dynamic attributes are used)
- Added Database class copy constructor and assignment operator
- Tango is now compiled with Debian hardenning flags on to add text
List of changes (bug fixes)

Bug fixes

---

Bug recorded in sourceForge:

- 3129849 : TANGO_HOST case sensitive for some event usage
- 3151801 : Missing some attribute properties in UserDefaultAttrProp class
- 3165120 : Yet another type in doc
- 3206916 : Another type in doc
- 3213730 : Device server add wrong ',0' in attribute abs_change property
- 3259442 : Macos compilation on x86
- 3267364 : Typo in documentation
- 3277453 : Database class and Tango Access Control
- 3280851 : Wrong state computation
- 3285370 : Printing operator for DeviceData class
- 3285372 : Wrong lock removal of last locked device from a locking thread (Windows specific)
- 3285674 : NaN in write_attribute() call (With a control system prop. to allow/disallow NaN)
- 3313211 : Polling threads pool management
- 339975 : ULong data type and memorized writable attribute
- 3400550 : State computation with alarmed attributes
- 3413944 : Memorized attribute written at init
- 3468928 : Does not compile with gcc 3.3
- 3480524 : Wrong printed date (and reported in blackbox) when used on 64 bits computer to add text

Other bugs

- When user pushes event, pushes first event when it is initied (when the event detection is done by the lib)
- In case of consecutive signal installations and removals.
- Bug in error message and in inserters in DbDatum class for unsigned char data type
- Bug when updating database due to one attribute configuration change
- Bug when using the WAttribute::set_min_value() methods family: The attribute was not flagged as attribute with minimun value defined
- Doc: Fix bug in Database::get_device_attribute_property() method usage example
- Bug in WAttribute::set_min_value() and Wattribute::set_max_value() methods for unsigned char data type.
The data was stored in database as ascii characters
- Device server crashes when you kill it if there are some long running actions when the signal is received.
- It's now possible to define in code that state and status has to be polled
- It's now possible to define an archive event period or a periodic event period for state or status attributes
- Possible device server process crash (depending how you are lucky) when trying to start one with an instance name not defined in database
- Bug when reading attribute from CACHE when the attribute is not polled. The returned exception was not correct
- Wrong printed date (and reported in blackbox) when used on 64 bits computer to add text
Compatibility - Compilation

• Major release
  • Recompile all objects files belonging to the same process
  • Network compatible

• Compilation
  • Use ZMQ library
    • ZMQ include files
    • `-lzmq` in linker command line
  • Use some C++11 features when available (gcc >= 4.3 or VC10)
    • `-std=c++0x` in compiler command line
    • Makefile generated by Pogo manage this
Mutable attribute properties

- Their value can be **modified by users**
- 20 properties concerned:

<table>
<thead>
<tr>
<th>label</th>
<th>min_value</th>
<th>abs_change</th>
<th>period</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>max_value</td>
<td>rel_change</td>
<td>archive_period</td>
</tr>
<tr>
<td>unit</td>
<td>min_alarm</td>
<td>arch_abs_change</td>
<td>delta_t</td>
</tr>
<tr>
<td>standard_unit</td>
<td>max_alarm</td>
<td>arch_rel_change</td>
<td></td>
</tr>
<tr>
<td>display_unit</td>
<td>min_warning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>format</td>
<td>max_warning</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>delta_val</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **not concerned:** eg. name, data_type, data_format, max_x, max_y

- Attribute's type:
  - string
  - attribute's type
  - DevDouble
  - DevLong
What has changed?

- setting / getting properties
  - new setters and getters on the server side
  - set / get all mutable properties in one go on the server side
  - validity checks
- setting user default values
- restoration of default values
- database optimization
- attribute construction exception handling
- templates
Set / get properties on the server side

• set / get all mutable properties in one go
  • same functionality as the network call: set_attribute_config()
    1. get attribute configuration
    2. modify some properties
    3. set attribute configuration

• overloaded methods
  Attribute::set_properties() and Attribute::get_properties()

• new template class MultiAttrProp<T> as a properties' values carrier
  • properties' values can be provided as strings or numerical data

• performs validity checks
  • eg. min_value < (?) max_value
  • delta_val = “123abe”

• rollback mechanism
Set / get properties on the server side example

network call:

```java
DeviceProxy device = new DeviceProxy("DevName");

AttributeInfo ai;
ai = device->get_attribute_config("AttName");
ai.min_alarm = "1.2";

AttributeInfoList ai_list;
ai_list.push_back(ai);
device->set_attribute_config(ai_list);
```

server side:

```java
MultiAttribute *attributes = this->get_device_attr();
Attribute &attr = attributes->get_attr_by_name("AttrName");

MultiAttrProp<DevDouble> multi_prop;
attr.get_properties(multi_prop);
DevDouble alarm_val = 1.2;
multi_prop.min_alarm = alarm_val;
// or multi_prop.min_alarm = "1.2";

attr.set_properties(multi_prop);
```
Set / get properties on the server side

• new set / get template methods:

  • Attribute::set_min_alarm(T &) & Attribute::get_min_alarm(T &)
  • Attribute::set_max_alarm(T &) & Attribute::get_max_alarm(T &)
  • Attribute::set_min_warning(T &) & Attribute::get_min_warning(T &)
  • Attribute::set_max_warning(T &) & Attribute::get_max_warning(T &)

• accept both string and numerical values
• validity checks are performed
Reset attribute properties to default values

• 3 levels of default values:
  • library defaults
  • user defined defaults
  • class level defaults

• keywords:
  • “Not specified” - unconditionally restore library defaults
  • “” (empty string) – restore user defaults, if not defined bring library defaults
  • “NaN” - restore class defaults, if not defined bring user defaults, if no user defaults defined, reset to library defaults
Reset attribute properties to default values example

<table>
<thead>
<tr>
<th></th>
<th>class defaults</th>
<th>user defaults</th>
<th>library defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Not specified”</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>“” (empty string)</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>“NaN”</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
</tbody>
</table>
Attribute constructor exception handling

- in Tango 8 attribute properties **validity** has been **reinforced**
  - \( \text{min\_alarm} < \text{max\_alarm} \) (etc.)
  - no letters if numerical value expected
  - **RDS** alarm properly defined (both \( \text{delta\_t} \) & \( \text{delta\_val} \) set)

- exception may occur at the device server startup if forbidden property values are stored in the database
  - all raised exceptions are stored locally
  - reading & writing a value of the attribute is refused
    - exceptions list is thrown
    - users must modify the corrupted properties
    - if configuration valid – allow read & write
Attribute constructor exception handling example

User

- read StepMotorPosition
  - exception! min_alarm > max_alarm; period NaN
- set max_alarm = 5
- read StepMotorPosition
  - exception! period NaN
- set period = 1000
- read StepMotorPosition
  - StepMotorPosition = 3

DS

- set max_alarm = 5
- set period = 1000

DB

- StepMotorPosition:
  - min_alarm 2
  - max_alarm 0
  - period 123abc
  - value 3

- min_alarm 2
- max_alarm 5
- period 123abc
- value 3

- min_alarm 2
- max_alarm 5
- period 1000
- value 3
New event system (part 1 – Everything is fine)
New event system

- Tango user point of view
  - No change at all in method calls (both on client and server side)
  - Filters not available any more (no answer on mailing list – 29/06/2011)
  - New DeviceProxy::subscribe_event() methods family without this parameter
    - The old ones still work
- Tango CS administrator
  - Notifd not needed any more
    - IF both client AND server use Tango 8
- Tango kernel
  - Many changes!
ZMQ

- A layer to build distributed system
  - Between threads within a process
  - Between processes within a host
  - Between hosts
- Supports several communication patterns
  - Request/Reply, Publish/Subscribe, Push/Pull,...
- Only takes care of transporting data
  - No encoding provided
- Written in C but many bindings available
  - C++, Java, Python, Erlang, Ruby,...
The basics

• Two main points
  • 1 - Use Publish / Subscribe pattern
    • The publisher is the DS
    • The subscribers are the applications
  • 2 – Use CORBA CDR (marshalling - unmarshalling) to encode / decode data
    • Same structures than those defined in the CORBA IDL Tango file
Transported data

• Use ZMQ Multipart message

• On the wire, one event is a 4 parts message:
  • Part 1: The event FQDN (string – lower case)
    • tango://kidiboo:10000/et/test/01/current.change
  • Part 2: The endianess (One byte)
    • 0 = big endian, 1 = small endian
  • Part 3: Object selection (structure – Encoded using CORBA CDR)
    • Method name (string – lower case) – Not used yet
    • Global object identifier (bytes sequence) – Not used yet
    • Version
    • Counter
    • Exception flag
Transported data

- Part 4 – Event data (structure – Encoded using CORBA CDR)
- Use structure defined in Tango IDL
Publisher / Application side event filtering

• How many ZMQ publisher sockets per DS (Splitting events on publishers) ?
  • 1 per DS → All events for all DS devices sent to the application!!
    • ZMQ layer in the application will do the filtering
  • 1 per device and event type → Many publishers (3 fd / publisher)
    • No filtering needed on applications
  • 4 publishers / device
    • 1 pub/change + 1 pub/archive + 1 pub/periodic
    • 1 pub/remaining event (att conf change, data ready, user) → Some filtering needed
    • 1 specific publisher for heartbeat event
  • Example:
    • 1 DS with 20 devices → 81 (1 + 4*20) publishers (243 fd)
Publisher / Application side event filtering

• ZMQ release 3 offers “subscription forwarding”
  • ZMQ filtering done on the first X bytes of the transported data
  • Filtering done on the publisher side (DS side)

• Using ZMQ 3
  • 2 publishers:
    • 1 dedicated to the DS heartbeat event
    • 1 dedicated to all events for all devices embedded in the DS
Establishing event connection

- Subscriber (appli) needs the publisher (DS) host IP address and the selected port number (the ZMQ endpoint)

- A new DS admin device cmd: ZMQEventSubscriptionChange
  - Same inputs than the actual EventSubscriptionChange cmd
  - Event name
  - Out = DevVarLongStringArray data type
    - Out string[0] = DS heartbeat ZMQ publisher endpoint
    - Out string[1] = Event ZMQ publisher endpoint
    - Out long[0] = Tango lib release number
    - Out long[1] = Device IDL release

- No need to store the endpoint in database
  - This feature is not available for ZMQ event
Event Compatibility

• Both event systems (notifd / ZMQ) in Tango 8

<table>
<thead>
<tr>
<th>Appli</th>
<th>Device Server</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tango 7</td>
<td>Tango 7</td>
<td>OK (notifd)</td>
</tr>
<tr>
<td>Tango 8</td>
<td>Case 2 (notifd)</td>
<td>OK (ZMQ) **</td>
</tr>
</tbody>
</table>

• Case 1
  • Appli uses admin device EventSubscriptionChange cmd → Old appli → Use notifd

• Case 2
  • Appli uses admin device ZMQEventSubscriptionChange cmd → Exception → appli uses EventSubscriptionChange → Server uses notifd

• ** : Only if device(s) inherit from Device_4Impl. Otherwise, notifd
Events and threads

• A Tango 8 DS has at least 8 threads
  • Main thread
  • 3 ORB's threads
  • Signal thread
  • Heartbeat thread
  • 2 ZMQs threads
  • X > 0 threads for polling thread pool
  • Y > 0 threads for requests service
Events and threads

• A Tango 8 client using events has at least 6 threads
  • Main thread
  • One ORB thread
  • 2 ZMQs thread
  • 2 Tango event system threads (KeepAlive and EventConsumer)

• Callback execution on client side is single-threaded
  • Could be changed if required (thread pool)
Performances

• Device server
  • Core 2 Duo 2.66 Ghz – 4 GB ram – 100 Mbit/sec – Ubuntu 11.10
• Client
  • P4 2.4 Ghz – 1.5 GB ram – 100 Mbit/sec – Ubuntu 10.10

<table>
<thead>
<tr>
<th></th>
<th>1 DevLong</th>
<th></th>
<th>1 K DevLong</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tango 7</td>
<td>Tango 8</td>
<td>Tango 7</td>
<td>Tango 8</td>
</tr>
<tr>
<td>1</td>
<td>770</td>
<td>25000</td>
<td>650</td>
<td>2100</td>
</tr>
<tr>
<td>2</td>
<td>770</td>
<td>13000</td>
<td>460</td>
<td>1200</td>
</tr>
<tr>
<td>5</td>
<td>400</td>
<td>5400</td>
<td>200</td>
<td>540</td>
</tr>
<tr>
<td>10</td>
<td>220</td>
<td>2700</td>
<td>100</td>
<td>270</td>
</tr>
</tbody>
</table>
New Event system – Part 2: Weather turns bad...
Events and multicasting

- ZMQ implement pub/sub with multicasting using OpenPGM
  - Implementation of the PGM protocol
- Compile ZMQ with the "--with-pgm" option
- Multicasting is more tricky to set-up due to buffer tuning and rate limited protocol (PGM)
- Not used in Tango 8: Unicast is the default
  - Tango 8.1 will add multicasting
  - A CtrlSystem property will allow the CS administrator to define
    - which event(s) has to be propagated using multicast
    - using which multicast group
Events and HWM

- HWM = ZMQ buffers High Water Mark
  - Max number of events in the ZMQ buffer
  - When full, ZMQ discards event without reporting errors
Events and HWM

- Library set a default value of 1000 for both servers/clients
- Control system properties belonging to the CtrlSystem free object
  - DSEventBufferHwm
  - EventBufferHwm
- At client or device server level using library calls
  - Util::set_ds_event_buffer_hwm()
  - ApiUtil::set_event_buffer_hwm()
- Using environment variables
  - TANGO_DS_EVENT_BUFFER_HWM
  - TANGO_EVENT_BUFFER_HWM
Events and HWM

• ZMQ drops events:
  • Event counter in the third part of the event data transferred on the wire
  • If missing event(s)
    • Callback called with error flag set
New event system – Part 3: Things turn bad
ZMQ?

- ZMQ 3 selected in June 2011
  - Still not the "stable release"
  - Still have 3 bugs not solved "Critical issues"
    - First-part of multipart message lost
    - Loosing multi-part message when using OpenPGM
    - HWM management on publisher side
  - ZMQ 3 not wire compatible with ZMQ 2
  - Java bindings not available for 3.1 because unstable!
    - ZMQ events between C++ processes
    - TangORB developed for ZMQ with a ZMQ 2 test device server
  - Tango 8 at ESRF only uses events between C++ processes
    - DS and archiving systems (500 attributes stored using ZMQ)
    - Events between device servers
ZMQ or Crossroads-io ?

- Main ZMQ developers have forked ZMQ
  - Crossroads-io (http://www.crossroads.io/)
  - Implement new features (socket disconnection)
  - Wire compatible with ZMQ 2 (not ZMQ 3)!
  - Today it is still release 1.1 (brand new)
  - Community much smaller than ZMQ but more active
- Too early to move to Crossroads-io but it's something which has to be followed and which may happen!
  - Compatibility problems !!
Future?

- It's not that bad. Several possible ways to deal with this situation thanks to the event system re-factoring done in Tango 8
  - ZMQ progress well
    - Continue to use it
  - Crossroads-io is more sexy in several months
    - Move to crossroads-io
      - With or without compatibility with processes using ZMQ
        - We are doing this kind of compatibility between ZMQ and notifd events
        - Should not be too difficult
  - Both of them disappears!!
    - Replace the Event transport layer by Tango group
- We will never return to the use of external process like notifd
Miscellaneous new features

• Polling in Tango class:
  • New set of methods in DeviceImpl class to manage polling in your Tango class code
    • is_attribute_polled(), is_command_polled(),
      get_attribute_poll_period(), get_command_poll_period(),
      poll_attribute(), poll_command(), stop_poll_attribute(),
      stop_poll_command()

• C++11 (When available):
  • Move constructor and assignment operator for DeviceData and DeviceAttribute classes
    • Copy constructor and assignment operator really copy the data
Some bug fixes

- SF bug 3285674: NaN in write_attribute()
  - A new control system property to allow/disallow NaN
    - CtrlSystem/WAttrNaNAllowed
    - Disable by default
- SF bug 3399975: Memorized attributes
  - All data types supported
- State and Status polling can be defined in code (Pogo) like any other attributes
- Now possible to define archive or periodic event period for State and Status
Distributions

- Will be Tango 8.0.5
- Update of all included packages
  - Database server AND its stored procedure (Release 1.8 - Update it as well)
  - Jive, Pogo, Astor
  - ATK
  - …
- Windows
  - Win32 / VC9
  - Win64 / VC10
- Debian (Fred)
  - What about ZMQ 3.1?
Pogo

• Support Tango 8
  • State/status with polling period in code
  • All attribute properties are now managed
  • Better Tango class inheritance using Tango 8 new methods
    \( \text{Util}::\text{is}_{\_}\text{svr}\_\text{starting}(), \text{Util}::\text{is}_{\_}\text{device}\_\text{restarting}() \)
Pogo

- Support dynamic attributes (ESRF way)
** Read BeamLineState attribute
* Description: 
* Data type: Tango::DevState
* Attr type: Scalar
*/

void BLsurvey::read_BeamLineState(Tango::Attribute &attr)
{
    DEBUG_STREAM << "BLsurvey::read_BeamLineState(Tango::Attribute &attr) entering... " << endl;
    Tango::DevState *att_value = get_BeamLineState_data_ptr(attr.get_name());

    /*----- PROTECTED REGION ID(BLsurvey::read_BeamLineState) ENABLED START -----*/

    // Set the attribute value
    for (unsigned int i=0 ; i<beamLines.size() ; i++)
    {
        string attName = beamLines[i]->name;
        attName += "State";
        if (attName==attr.get_name())
        {
            *att_value = beamLines[i]->getState();
            attr.set_value(*att_value);
        }
    }

    /*----- PROTECTED REGION END -----*/ // BLsurvey::read_BeamLineState
}

void BLsurvey::add_dynamic_attributes()
{
    // Example to add dynamic attribute:
    // add_BeamLineState_dynamic_attribute("MyAttribute");

    /*----- PROTECTED REGION ID(BLsurvey::add_dynamic_attributes) ENABLED START -----*/

    for (unsigned int i=0 ; i<beamLines.size() ; i++)
    {
        string attStateName = beamLines[i]->name;
        attStateName += "State";
        add_BeamLineState_dynamic_attribute(attStateName);
    }

    /*----- PROTECTED REGION END -----*/ // BLsurvey::add_dynamic_attributes()
Java device server (From Gwenaelle - Soleil)

• Work done by Soleil
  • A beta release will soon be available
    • Without event
  • Downloadable from the pink site, documentation will also be available on the pink site
  • Acceptance test: The C++ Tango test suite should work on a Java device server (except event part)
    • Well advanced
  • Event will be added by ESRF when Java binding for ZMQ 3 will be ready