# Tango Request For Comments (RFC) Documentation

***Release 9.3***

**Tango Controls Community**

**Apr 22, 2021**

# CONTENTS

CHAPTER

ONE

# TANGO REQUEST FOR COMMENTS (RFC)

This repository is the home of all Tango Open Specification.

## 1.1 Mission

The goal of this RFC project is to provide a formal specification of the current (V9 LTS) Tango Controls system. This specification shall include:

1. concepts,
2. terminology,
3. protocol behaviour,
4. conventions,

each on a sufficient level for future evolution of Tango Controls and/or implementation in other languages. In that respect, concepts are more important than implementation details.

## 1.2 Contribution

The process to add or change an RFC is the following:

- An RFC is created and modified by pull requests according to the Collective Code Construction Contract (C4).
- The RFC life-cycle SHOULD follow the life-cycle defined in the Consensus-Oriented Specification System (COSS).

Read more here.

Here is the list of contributors.

## 1.3 RFCs

The table below summarises all available or expected specifications. For the current "work in progress status", please check either PRs or Wiki.

| Short Name | Title | Status | Editor |
|---|---|---|---|
| RFC-1 | The Tango control system | Raw | Lorenzo Pivetta |
| *RFC-2* | The device object model | Draft | Vincent Hardion |
| *RFC-3* | The command model | Draft | Sergi Blanchi-Torné |
| *RFC-4* | The attribute model | Draft | Sergi Blanchi-Torné |
| *RFC-5* | The property model | Draft | Gwenaelle Abeillé |
| *RFC-6* | The database system | Draft | Gwenaelle Abeillé |
| *RFC-7* | The pipe model | Draft | Reynald Bourtembourg |
| *RFC-8* | The server model | Draft | Lorenzo Pivetta |
| *RFC-9* | Data types | Draft | Gwenaelle Abeillé |
| RFC-10 | The Request-Reply protocol | Raw | Reynald Bourtembourg |
| RFC-11 | The Request-Reply protocol - CORBA implementation | Raw | |
| *RFC-12* | The Publisher-Subscriber protocol | Draft | Vincent Hardion |
| RFC-13 | The Publisher-Subscriber protocol - ZeroMQ implementation | Raw | |
| *RFC-14* | Logging service | Raw | Sergi Blanchi-Torné |
| *RFC-15* | The dynamic attribute and command | Draft | Reynald Bourtembourg |
| RFC-16 | Cache system | Raw | |
| RFC-17 | Memorised attribute service | Raw | |
| RFC-18 | Authorisation system | Raw | |
| RFC-XX | High Level API | Raw | |
| RFC-XX | High Level API - Python implementation | Raw | |
| RFC-XX | High Level API - Java implementation | Raw | |

CHAPTER

TWO

# 1/TANGO CONTROL SYSTEM

This document describes Tango, a control system framework. This RFC define the specification of a control system following the philosophy of Tango.

See also: Y/OtherTemplate

## 2.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 2.2 Tango Specification

The TANGO control system is a device-oriented controls toolkit for controlling any kind of hardware or software and building SCADA systems.

### 2.2.1 Goals

Tango aims to

The software used in large facilities can be compared with the software for stock markets – huge amount of data must be displayed on the monitor in a real time and being processed and being saved in databases for post processing.

So, the toolkit was mainly developed for research facilities needs, but the idea and concept (philosophy) behind was to create a framework.

Additionally, it aims to:

- Provide ...

- Be usable as . . .
- Be compatible . . .

### 2.2.2 Use Cases

There are X main use cases for Template:

- To reduce the entry cost of RFC submission.
- To make coherent the RFC

. . .

CHAPTER

THREE

# 2/DEVICE MODEL

This document describes the Tango Device model specification version 5.0.

See also: *1/TANGO*, *3/Command*, *4/Attribute*, *5/Property*, *6/Database*, *7/Pipe*, *8/Server*

## 3.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 3.2 Tango Device Specification

A Device is designed to represent any controlled object in Tango Controls System. This specification is inspired by the comment written by A. Götz and E. Taurel in the tango.idl file from the original implementation (https://github.com/tango-controls/tango-idl)

### 3.2.1 Goals

The Tango Device represents the fundamental interface for all TANGO objects. Directly inspired by object-oriented programming, a Device object has data and actions. It allows users of the Tango Controls System to access (read and/or write) information (data) stored in or processed by hardware or virtual devices and call actions which may influence hardware state and behaviour. Typically, access to data is provided via Device's Attributes and Pipes and actions are initiated by calling Device's Commands.

Additionally, it aims to:

- Provide a standard way of translating different communication protocols to a communication protocol implemented by Tango Controls. For example, the same generic GUI application may be used to control a power supply interfaced with the SCPI protocol and to control a stepper motor driver interfaced with the CAN bus.

- Be usable as a logical abstraction of the hardware. For example, the same GUI application may be used to control power supplies which are powering magnets and powering heat coils.
- Expose, to Users and other Actors of the Tango Controls System, a semantic interface which is well understood, intuitive, and match the object-oriented paradigm. For example, a power supply Device may be implemented in Java, Python or C++ language. Irrespectively to the implementation language, Attributes *current* and *voltage* can be implemented as field members of an object, while *on* and *off* commands can be implemented as methods of the same object.

### 3.2.2 Use Cases

Typical use cases for Device are:

- It is common for particles accelerator systems to use a set of power supplies for powering a magnet system. Power supplies may come from different vendors. These power supplies may use different communication interfaces and protocols for a remote control. However, there is a need to steer these power supplies from a remote locations in a unified and consistent way. Steering includes setting and reading of voltages and currents as well as switching the power supplies on and off, each individually. Having them integrated into a system as Devices, each with an unique name, with Current and Voltage attributes and On() and Off() commands fulfills the requirement and assure consistency.
- Remote monitoring of a complex device like an electro-magnet, which is powered by power supplies and requires a cooling system may be provided by a Device with the following attributes: MagneticField, CoilTemperature, State and Status and with the commands On() and Off(). This Device, to serve its interface is using information coming (directly or through other devices) from power supplies and cooling systems.

## 3.3 Specification

A Tango Device is a strict definition of a distributed object.

- configuration is represented in the form of Properties.
- data are represented in the form of Attributes and Pipes.
- actions are represented in the form of Commands.

Its model can be represented as a defined tree which each element are from a defined type: Class, Device, Property, Attribute, Pipe, Command following the rules below:

- The Device is a distributed object which SHALL be accessible locally or via the network.
- The Device SHALL be an instance of one Class, see *Device Class section*.
- The Device MAY have one or several Property, called Device Property, see *RFC 5/Property*.
- The Device MAY have one or several Attribute, see *RFC 4/Attribute*.
- The Device MAY have one or several Pipe, see *RFC 7/Pipe*.
- The Device MAY have one or several Command, see *RFC 3/Command*.
- The Device SHALL have one Init Command.
- The Device SHALL have one State Attribute.
- The Device SHALL have one Status Attribute.
- The Device SHALL have one State Command.
- The Device SHALL have one Status Command.

**Note**: Future specification may remove 'SHALL have' requirements for State and Status Commands.

- The Device SHALL have one unique identifier which represents its Device Name.

### 3.3.1 Naming convention

- The Device Name SHALL use the following convention:

```
device-name = domain "/" family "/" member
domain = 1*VCHAR
family = 1*VCHAR
member = 1*VCHAR
```

### 3.3.2 Device Class

A Class is an association of a list of Class Properties, a list of Attributes, a list of Pipes, a list of Commands, a list of Device Properties with a Device Class Name.

- The Class Name SHOULD reflect its instances application context.
- The Class Name SHALL use the following convention:

```
class-name = 1*VCHAR
```

- The Class MAY have one or more Device instance.
- The Class MAY have one or several Property, called Class Property.
- The Device SHALL have at least all Attributes, Pipes and Commands defined by its Class.
- The Device MAY have Attributes and/or Commands not defined by its Class. These are called Dynamic Attributes and/or Dynamic Commands respectively, see *RFC-15*.
- Dynamic Attributes and/or Dynamic Commands MAY be added to the Device during transition from the phase Not Exported to the phase Exported.
- In the Exported phase, Dynamic Attributes and/or Dynamic Commands MAY be added to the Device.
- In the Exported phase, Dynamic Attributes and/or Dynamic Commands MAY be removed from the Device.

### 3.3.3 Device lifecycle

The Device lifecycle is:

- Device MUST be in one of two phases: Exported or Not Exported.
- In the Exported phase, Device MUST be accessible to actors of the Tango Controls System.
- In the Not Exported phase, Device MUST NOT be accessible to actors of the Tango Controls System.
- At the transition from the phase Not Exported to the phase Exported the Device MUST be initialised as if its Init Command was executed.
- At the transition from the phase Exported to the phase Not Exported the Device SHOULD gracefully be uninitialised.
- The Device Server MUST manage Device lifecycle.

### 3.3.4 Init Command

The Init Command purpose is to re-initialise the Device.

- The Init Command, when called, SHALL in a sequence:
  - gracefully un-initialise the Device returning to a state before its first initialisation,
  - initialise the Device using the values of its Properties.

### 3.3.5 Device State and Status

The State Attribute represents Device State.

- The *data type* of State Attribute MUST be `DevState`. The *read value* of Status Attribute MUST be of `DevState` data type.
- The State Attribute's *writable* metadata MUST be `READ`.
- The *data format* of State Attribute MUST be `SCALAR`.
- The *read value* of State Attribute SHOULD reflect the context of a running Device. If Device is related to hardware or a virtual entity or some process, the value of State Attribute SHOULD reflect the state of the entity or process.
- The *data type* of Status Attribute MUST `DevString`. The *read value* of Status Attribute MUST be of `DevString` data type.
- The Status Attribute's `writable` metadata MUST be `READ`.
- The `data format` of the Status Attribute MUST be `SCALAR`.
- The Status Attribute SHOULD provide status information related to the Device.
- By default Status Attribute SHOULD indicate the Device State.
- The State Command MUST return the same value as would be read from the State Attribute at the time of the command execution.
- The Status Command MUST return the same value as would be read from the Status Attribute at the time of the command execution.

### 3.3.6 State Machine

- By default State Attribute value SHALL be `UNKNOWN`.
- By default the Device State SHOULD be in ALARM if at a given time the Device State is ON and at least one of Device Attributes has the quality set to ALARM or WARNING.

Value of Device's State Attribute MAY impact how the Device responds to:

- Reading and writing of its Attribute or Pipe,
- Calling a Command.

The Device MAY not execute (block) the Attribute or Pipe read or write operation or the command execution code for individual States. In such a case the Device SHALL respond with throwing a `DevFailed` exception. The way how a Device responds to the above calls define a State Machine.

- A Device MAY use a State Machine.

### 3.3.7 Reserved Device Names

However it is possible to use any Device Name as it is stated in *Naming convention section*, some groups of names are reserved. One is Admin Device Name (see *RFC 8/Server*), defined as follows:

```
admin-device-name = domain "/" family "/" member
domain = %i"dserver"
family = 1*VCHAR
member = 1*VCHAR
```

`admin-device-name` MUST not be used for Devices of other than `DServer` Device Class.

### 3.3.8 Reserved Class Names

Devices of some of Classes are used to provide standard Tango Controls System services. These Classes' Names SHOULD not be used for other purposes. Below is a list of reserved Class Names:

- `%i"DataBase"`, see *RFC 6/Database*
- `%i"TangoAccessControl"`
- `%i"DServer"`

CHAPTER

FOUR

# 3/COMMAND

This document describes the specification of the Tango Command model, a guideline for the implementer of the Tango Control System.

See also: *2/Device*, *4/Attribute*

## 4.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 4.2 Tango Command Specification

The specification of Command aims to define a standard structure and type to use in any implementation of Tango.

### 4.2.1 Goals

A Command in a Tango Controls represents an action, a sequence or other procedure executed remotely by a Device. The meaning of commands is quite similar to the meaning of methods in object oriented design.

This specification aims to define a Command in order to be able to :

- Execute action considering an argument
- Execute an asynchronous action
- Expect a reply in any case within a certain timeout
- Be represented by meta data

Additionally, Command may be used for:

- Reading and writing device parameters, as an alternative to read and write Tango Attribute for legacy reason (although Tango Attribute is the prefer way).

### 4.2.2 Use Cases

The use of Command by the developer of Device allows to resolve different situation:

- A command "Stop" can trigger the ending of a movement for a motor or an integration for a detector
- A command "Send" can take any SCPI command as argument and wait for the reply
- A command "Execute" can launch a macro which may take long time to be executed in the background. The reply is just an acknowledgement that the sequence will be executed. The current state of progress can be retrieve by the state of the Tango Device

## 4.3 Specification

A Tango Command is a strict definition of a function apply at the Device context:

- The Command SHALL have a unique identifier, called the Command Name
- The Command input or output are called Argument
- The Command SHALL have one input Argument, called ARGIN
- The Command SHALL have one output Argument, called ARGOUT

An Argument represents the input and output of a Command and its specification can only applied to a Command as below:

- Argument SHALL have an Argument Type which set the type of the Argument Value
- Argument MAY have an Argument Description which describe the meaning of the Argument Value

An Argument Value only exists during the phase of execution of a Command. In this case a Command MAY have 2 Arguments Values, one for ARGIN and one for ARGOUT.

The Argument Type, Argument Value, Argument Representation and Argument Description are specified below as litteral representation. This SHOULD NOT constraint the implementation to use a binary format for the transport layer.

```
argument = dev-void | dev-double | dev-boolean | dev-float | dev-short | dev-long |␣
↪dev-long64 | dev-string | dev-uchar | dev-ushort | dev-ulong | dev-ulong64 | dev-
↪boolean-array | dev-double-array | dev-float-array | dev-short-array | dev-long-
↪array | dev-long64-array | dev-char-array | dev-string-array | dev-ushort-array |␣
↪dev-ulong-array | dev-ulong64-array | dev-long-string-array | dev-double-string-
↪array | dev-boolean-array | dev-encoded | dev-encoded-array

argument-desc = 0*CHAR

; Float Definition inspired by python float definition
; Some examples of floating point literals: 3.14 10. .001 1e100 3.14e-10
floatnumber = [ "-" ] pointfloat | exponentfloat
pointfloat = [intpart] fraction | intpart "."
exponentfloat:  ( %d1-9 DIGIT* | pointfloat) exponent
intpart:        %d1-9 DIGIT* | %d0
fraction:       "." DIGIT+
exponent:       ("e"|"E") ["+"|"-"] DIGIT+
```

(continues on next page)

(continued from previous page)

```
dev-void = dev-void-type WSP dev-void-value
dev-void-type = "DEVVOID" | "DevVoid"
dev-void-value = NULL

dev-double = dev-double-type WSP dev-double-value
dev-double-type = "DEVDOUBLE" | "DevDouble"
dev-double-value = floatnumber ; double-precision normalized numbers in IEC 60559

dev-boolean = dev-boolean-type WSP dev-boolean-value
dev-boolean-type = "DEVBOOLEAN" | "DevBoolean"
dev-boolean-value = BIT

dev-float = dev-float-type WSP dev-float-value
dev-float-type = "DEVFLOAT" | "DevFloat"
dev-float-value = floatnumber ; single-precision normalized numbers in IEC 60559

dev-short = dev-short-type WSP dev-short-value
dev-short-type = "DEVSHORT" | "DevShort"
dev-short-value = ["-"] DIGIT* ; In the range of "(2^15) - 1" to "-(2^15)".

dev-long = dev-long-type WSP dev-long-value
dev-long-type = "DEVLONG" | "DevLong"
dev-long-value = ["-"] DIGIT* ; In the range of "(2^31) - 1" to "-(2^31)".

dev-long64 = dev-long64-type WSP dev-long64-value
dev-long64-type = "DEVLONG64" | "DevLong64"
dev-long64-value = ["-"] DIGIT* ; In the range of "(2^63) - 1" to "-(2^63)".

dev-string = dev-string-type WSP dev-string-value
dev-string-type = "DEVSTRING" | "DevString"
dev-string-value = DQUOTE CHAR* DQUOTE

dev-uchar = dev-uchar-type WSP dev-uchar-value
dev-uchar-type = "DEVUCHAR" | "DevUChar"
dev-uchar-value = DIGIT* ; In the range of "0" to "(2^8) - 1". In abnf syntax it may␣
↪correspond to %d0-255

dev-ushort = dev-ushort-type WSP dev-ushort-value
dev-ushort-type = "DEVUSHORT" | "DevUShort"
dev-ushort-value = DIGIT* ; In the range of "0" to "(2^16) - 1".

dev-ulong = dev-ulong-type WSP dev-ulong-value
dev-ulong-type = "DEVULONG" | "DevULong"
dev-ulong-value = DIGIT* ; In the range of "0" to "(2^32) - 1".

dev-ulong64 = dev-ulong64-type WSP dev-ulong64-value
dev-ulong64-type = "DEVULONG64" | "DevULong64"
dev-ulong64-value = DIGIT* ; In the range of "0" to "(2^64) - 1".

dev-state = dev-state-type WSP dev-state-value
dev-state-type = "DEVSTATE" | "DevState"
dev-state-value = "ALARM" | "INSERT" | "STANDBY" | "CLOSE" | "MOVING" | "UNKNOWN" |
↪"DISABLE" | "OFF" | "EXTRACT" | "ON" | "FAULT" | "OPEN" | "INIT" | "RUNNING"

dev-boolean-array = dev-boolean-array-type  WSP dev-boolean-array-value
dev-boolean-array-type = "DEVVARBOOLEANARRAY" | "DevVarBooleanArray"
```

(continues on next page)

(continued from previous page)

```
dev-boolean-array-value = "[" dev-boolean [ "," dev-boolean)* ] "]"

dev-double-array = dev-double-array-type  WSP dev-double-array-value
dev-double-array-type = "DEVVARDOUBLEARRAY" | "DevVarDoubleArray"
dev-double-array-value = "[" dev-double [ "," dev-double)* ] "]"

dev-float-array = dev-float-array-type WSP dev-float-array-value
dev-float-array-type = "DEVVARFLOATARRAY" | "DevVarFloatArray"
dev-float-array-value = "[" dev-float [ "," dev-float)* ] "]"

dev-short-array = dev-short-array-type WSP dev-short-array-value
dev-short-array-type = "DEVVARSHORTARRAY" | "DevVarShortArray"
dev-short-array-value = "[" dev-short [ "," dev-short)* ] "]"

dev-long-array = dev-long-array-type WSP dev-long-array-value
dev-long-array-type = "DEVVARLONGARRAY" | "DevVarLongArray"
dev-long-array-value = "[" dev-long [ "," dev-long)* ] "]"

dev-long64-array = dev-long64-array-type WSP dev-long64-array-value
dev-long64-array-type = "DEVVARLONG64ARRAY" | "DevVarLong64Array"
dev-long64-array-value = "[" dev-long64 [ "," dev-long64)* ] "]"

dev-char-array = dev-char-array-type WSP dev-char-array-value
dev-char-array-type = "DEVVARCHARARRAY" | "DevVarCharArray"
dev-char-array-value = "[" dev-char [ "," dev-char)* ] "]"

dev-string-array = dev-string-array-type WSP dev-string-array-value
dev-string-array-type = "DEVVARSTRINGARRAY" | "DevVarStringArray"
dev-string-array-value = "[" dev-string [ "," dev-string)* ] "]"

dev-ushort-array = dev-ushort-array-type WSP dev-ushort-array-value
dev-ushort-array-type = "DEVVARUSHORTARRAY" | "DevVarUShortArray"
dev-ushort-array-value = "[" dev-ushort [ "," dev-ushort)* ] "]"

dev-ulong-array = dev-ulong-array-type WSP dev-ulong-array-value
dev-ulong-array-type = "DEVVARULONGARRAY" | "DevVarULongArray"
dev-ulong-array-value = "[" dev-ulong [ "," dev-ulong)* ] "]"

dev-ulong64-array = dev-ulong64-array-type WSP dev-ulong64-array-value
dev-ulong64-array-type = "DEVVARULONG64ARRAY" | "DevVarULong64Array"
dev-ulong64-array-value = "[" dev-ulong64 [ "," dev-ulong64)* ] "]"

dev-long-string-array = dev-long-string-array-type WSP dev-long-string-array-value
dev-long-string-array-type = "DEVVARLONGSTRINGARRAY" | "DevVarLongStringArray"
dev-long-string-array-value =  dev-long-string-array-lvalue  dev-long-string-array-
↪rvalue
dev-long-string-array-lvalue = dev-long-array
dev-long-string-array-rvalue = dev-string-array

dev-double-string-array = dev-double-string-array-type WSP dev-double-string-array-
↪value
dev-double-string-array-type = "DEVVARDOUBLESTRINGARRAY" | "DevVarDoubleStringArray"
dev-double-string-array-value =  dev-double-string-array-dvalue dev-double-string-
↪array-rvalue
dev-double-string-array-dvalue = dev-double-array
dev-double-string-array-rvalue = dev-string-array
```

(continues on next page)

(continued from previous page)

```
dev-encoded = dev-encoded-type WSP dev-encoded-value
dev-encoded-type = "DEVENCODED | "DevEncoded"
dev-encoded-value =  dev-encoded-encoded-format  dev-encoded-encoded-data
dev-encoded-encoded-format = dev-string
dev-encoded-encoded-data = dev-char-array

dev-encoded-array = dev-encoded-array-type WSP dev-encoded-array-value
dev-encoded-array-type = DEVVARENCODEDARRAY | DevVarEncodedArray
dev-encoded-array-value = "[" dev-encoded [ ("," dev-encoded)* ] "]"
```

Additionally, the Command can be represented by meta data:

- A Command SHALL have a visibility level for the user called Display Level, which MAY be taken in consideration in the User Interface.

```
display-level = "OPERATOR" | "EXPERT"
```

Note: The description of a command is given by both the description of ARGIN and ARGOUT. There is no Command Description as such.

### 4.3.1 Reserved Commands

Although the user can define any kind of commands, some specific commands are essential to the Tango Device to work properly. The Reserved Command MUST be define as the Command Name, the ARGIN and the ARGOUT as describe below:

| Reserved Command | Name | ARGIN | ARGOUT |
|---|---|---|---|
| State Command | “State” | DevVoid | DevState |
| Status Command | “Status” | DevVoid | DevString |
| Init Command | “Init” | DevVoid | DevVoid |

### 4.3.2 Global Behaviour

- The Command SHALL always return a result when executed even when the ARGOUT is from the type DevVoid. The only exception is a “Fire and Forget” which force to be executed in one way (flag for asynchronous command execution).
- The execution of the Command SHALL be unique
- The execution of the Command SHALL apply only in the context of a Device

### 4.3.3 Naming convention

- The Command Name SHALL use the following convention:

```
command-name = 1*VCHAR
```

CHAPTER

FIVE

# 4/ATTRIBUTE

This document describes the Tango Attribute model specification version 5.0.

## 5.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 5.2 Tango Attribute specification

This specification is intended to formally document the Tango Attribute static model. Runtime representation of Attribute in conforming Tango implementations MUST follow this specification.

### 5.2.1 Goals

An Attribute is a Tango concept representing a physical quantity of a device. The main purpose of an Attribute is to provide read and (optionally) write access to this quantity.

In object oriented terminology, the Attribute corresponds to an instance variable (also called a field or a member) of a Device object. See *RFC 2/Device* for the definition of the Device.

### 5.2.2 Use Cases

Some example use cases of an Attribute are:

- an Attribute can represent a position of a motor device,
- an Attribute can represent a temperature of a thermocouple device.

## 5.3 Specification

An Attribute has:

- A set of static metadata that constitute Attribute's definition,
- A set of dynamically configurable properties, e.g. for alarming purposes,
- A set of runtime parameters describing Attribute's value at given point in time.

### 5.3.1 Attribute definition

The static metadata is part of the Attribute definition. It MUST be defined before an Attribute is created by any Tango implementation and MUST NOT change at runtime after the Attribute is initialized.

An Attribute MUST have associated following static metadata:

- *name*, a string identifying the Attribute. It MUST be unique[1] among all Attributes of a particular device. See `<attribute-name>` specification below,
- *data type*, an enumeration describing the type of the data (*RFC 9/Data Types*),
- *data format*, an enumeration describing the dimension of the data (one of *SCALAR*, *SPECTRUM*, *IMAGE*),
- *writable*, an enumeration describing the write access to the Attribute (one of *READ*, *WRITE*, *READ_WRITE*, *READ_WITH_WRITE*).
  - An Attribute can be read from, if *writable* is one of *READ*, *READ_WRITE*, *READ_WITH_WRITE*.
  - An Attribute can be written to, if *writable* is one of *WRITE*, *READ_WRITE*.
- *display level*, an enumeration describing visibility level of the Attribute (one of *EXPERT*, *OPERATOR*).

  **Note:** Although it is possible to use a wide range of characters in the Attribute's name, it is RECOMMENDED to use only numbers, ASCII letters (upper- and lower-case) and an underscore (_) to ensure compatibility with various tools and client applications. Also note that the Attribute name SHOULD contain at least one letter and SHOULD NOT start with a digit.

If *data format* is *SPECTRUM*, an Attribute MUST have associated following additional static metadata:

- *max dim x*, an integer describing the maximum allowed number of data elements. The Attribute MUST be able to store up to *max dim x* data elements. It MUST be greater than 0.

If *data format* is *IMAGE*, an Attribute MUST have associated following additional static metadata:

- *max dim x*, an integer describing the maximum allowed number of data elements in dimension X. The Attribute MUST be able to store up to *max dim x* data elements in the X dimension. It MUST be greater than 0,
- *max dim y*, an integer describing the maximum allowed number of data elements in dimension Y. The Attribute MUST be able to store up to *max dim y* data elements in the Y dimension. It MUST be greater than 0.

If *writable* is *READ_WITH_WRITE*, an Attribute MUST have associated following additional static metadata:

[1] In current implementation it is possible to define more than one Attribute with the same name, but only one can be accessed externally.

- *writable attribute name*, a string describing the *name* of associated writable attribute. It MUST point to an attribute. The Attribute pointed to MUST have *writable* set to one of *WRITE*, *READ_WRITE*.

If *data type* is *ENUM*, an Attribute MAY have associated following additional static metadata:

- *enum labels*, a list of strings describing textual representation of enumerated values. It MAY be empty.

### Memorized attribute

An Attribute MUST have associated following static metadata:

- *memorized*, a flag describing whether *set value* is stored in the database and restored during Attribute initialization,
- *write hardware at init*, a flag describing whether the memorized *set value* is written to the Attribute during initialization or *init* call. It is effective only if *memorized* is also set.

A conforming implementation MUST allow to set the *memorized* flag and MUST support memorization for attributes which:

- *data format* is *SCALAR*,
- *writable* is *WRITE* or *READ_WRITE*,
- *data type* is not *STATE* or *ENCODED*.

If *memorized* is set and if database is being used, the *set value* MUST be persisted in the *__value* Attribute property upon each write to the Attribute.

If *memorized* is set and *__value* Attribute property is other than "Not used yet" (default):

- only during Attribute initialization (at startup):
  - *set value* MUST be set with a value stored in *__value* Attribute property,
- during Attribute initialization (at startup) or during an *init* command execution, if *write hardware at init* is also set:
  - value stored in *set value* MUST be written to the Attribute.

### Forwarded attribute

An Attribute can be a forwarded attribute. A forwarded Attribute MUST have associated *__root_att* Attribute property. It MUST be defined before attribute initialization at startup. It MUST NOT be changed in runtime. It MUST point to an existing attribute in another device.

Allowed syntax for *__root_att* is specified below as `<full-attribute-name>`.

Each read and write request to a forwarded Attribute MUST be passed to the Attribute pointed by *__root_att*.

Each change to properties or to other metadata associated with a forwarded Attribute MUST be reflected in the Attribute pointed by *__root_att*.

Each change to properties or to other metadata associated with the Attribute pointed by *__root_att* MUST be reflected in the forwarded Attribute.

### 5.3.2 Attribute properties

An Attribute MAY have associated dynamic metadata in form of Properties (see *RFC 5/Property*).

Below table summarizes common attribute properties used by Tango. Detailed description is provided later in this section.

| Property | Name in database | Type | Default value |
|---|---|---|---|
| *description* | `description` | string | Not specified / No description |
| *label* | `label` | string | Not specified / No label |
| *unit* | `unit` | string | Not specified / No unit / *empty* |
| *standard unit* | `standard_unit` | number | Not specified / No standard unit |
| *display unit* | `display_unit` | number | Not specified / No display unit |
| *format* | `format` | string | %6.2f (float), %d (integer), %s(string, enum), Not specified (state, encoded, boolean) |
| *min value* | `min_value` | number | Not specified |
| *max value* | `max_value` | number | Not specified |
| *min alarm* | `min_alarm` | number | Not specified |
| *max alarm* | `max_alarm` | number | Not specified |
| *min warning* | `min_warning` | number | Not specified |
| *max warning* | `max_warning` | number | Not specified |
| *delta val* | `delta_val` | number | Not specified |
| *delta t* | `delta_t` | number | Not specified / 0 |
| *rel change* | `rel_change` | number | Not specified |
| *abs change* | `abs_change` | number | Not specified |
| *archive rel change* | `archive_rel_change` | number | Not specified |
| *archive abs change* | `archive_abs_change` | number | Not specified |
| *period* | `period` | number | 1000 |
| *archive period* | `archive_period` | number | Not specified |
| *__value* | `__value` | string | Not used yet |
| *__root_att* | `__root_att` | string | Not defined |

> **Note:** Both `delta_val` and `delta_t` MUST be set to non-default values in order for RDS (*read different than set*) alarms to work.

General properties:

- *description*, providing textual information about the Attribute,

- *label*, providing textual label to use as Attribute's name,
- *unit*, providing textual representation of Attribute's unit,
- *standard unit*, describing the conversion factor to transform Attribute's value into S.I units. It MUST be a valid numerical value,
- *display unit*, describing the conversion factor to transform Attribute's value into value usable in GUIs. It MUST be a valid numerical value,
- *format*, describing how to convert Attribute's data to a textual representation. It MUST comply to `printf` format string specification,
- *min value*, describing the minimum value of Attribute's *set value*. It MUST be defined only for Attributes with numerical *data type*. It MUST be defined only for writable Attributes. If defined: it MUST be a valid numerical value, it MUST be lower than *max value* (if specified),
- *max value*, describing the maximum value of Attribute's *set value*. It MUST be defined only for Attributes with numerical *data type*. It MUST be defined only for writable Attributes. If defined: it MUST be a valid numerical value, it MUST be greater than *min value* (if specified).

Properties for alarming purposes:

- *min alarm*, describing the threshold value of Attribute's *read value* below which the attribute is considered as having alarm *quality*. It MUST be defined only for Attributes with numerical *data type*. If defined: it MUST be a valid numerical value, it MUST be lower than *max alarm* (if specified),
- max alarm*, describing the threshold value of Attribute's *read value* above which the attribute is considered as having alarm *quality*. It MUST be defined only for Attributes with numerical *data type*. If defined: it MUST be a valid numerical value, it MUST be greater than *min alarm* (if specified),
- *min warning*, describing the threshold value of Attribute's *read value* below which the attribute is considered as having warning *quality*. It MUST be defined only for Attributes with numerical *data type*. If defined: it MUST be a valid numerical value, it MUST be lower than *max warning* (if specified),
- *max warning*, describing the threshold value of Attribute's *read value* above which the attribute is considered as having warning *quality*. It MUST be defined only for Attributes with numerical *data type*. If defined: it MUST be a valid numerical value, it MUST be greater than *min warning* (if specified),
- *delta val*, describing the maximum difference between Attribute's *read value* and *set value* after *delta t* time period, above which the attribute is considered as having alarm *quality*. It MUST be defined only for writable Attributes,
- *delta t*, describing the time period (in milliseconds) after which the difference between *read value* and *set value* must be lower than *delta val*. Otherwise the attribute is considered as having alarm *quality*. It MUST be defined only for writable Attributes, It MUST be defined if *delta val* is also defined.

Properties for event reporting purposes:

- *rel change*, *abs change*, *archive rel change*, *archive abs change*, describing the threshold values for relative and absolute change in Attribute's *read value* above which a *CHANGE* event or *ARCHIVE* event MUST be reported. It MUST be either a valid numerical value or a pair of valid numerical values separated by a `,`. If one value is specified, it MUST be used for both negative and positive change. If two values are specified, the first MUST be used for negative change and the second MUST be used for positive change. It MUST be defined only for Attributes with numerical *data type*,
- *period*, *archive period*, describing the minimum time period in milliseconds after which a *PERIODIC* or *ARCHIVE* event MUST be reported, regardless of Attribute's *read value*. If *period* is not specified, a default value of 1000 ms is used,

If relative change is specified, it MUST be expressed as a percentage change relative to the value reported in the previous event, e.g. a relative change of 1 will generate an event when:

```
(current value - previous value) / previous value > 1%
```

See *Attribute events section* for more details.

```
rel-change = change
abs-change = change

archive-rel-change = change
archive-abs-change = change

number = [ "-" ] 1*DIGIT [ "." ] *DIGIT
change = number [ "," number ]

period = 1*DIGIT
archive-period = period
```

### 5.3.3 Attribute runtime parameters

At given point in time an Attribute MUST have associated:

- *quality*, an enumeration describing the state *read value* (one of *VALID*, *INVALID*, *ALARM*, *CHANGING*, *WARNING*),
- *read value*, an object representing the value of the Attribute. It MUST conform to *data format* and *data type*,
- *read dim x*, an integer describing the number of data elements in *read value* in X dimension. If *data format* is *SCALAR*, it MUST be either 0 or 1. If *data format* is *SPECTRUM* or *IMAGE*, it MUST be between 0 and *max dim x*.
- *read dim y*, an integer describing the number of data elements in *read value* in Y dimension. If *data format* is *SCALAR* or *SPECTRUM*, it MUST be 0. If *data format* is *IMAGE*, it MUST be between 0 and *max dim y*.

Additionally, if *writable* is *WRITE* or *READ_WRITE*, at given point in time an Attribute MUST have associated:

- *set value*, an object representing the value set to the Attribute. It MUST conform to *data format* and *data type*,
- *write dim x*, an integer describing the number of data elements in *set value* in X dimension,
- *write dim y*, an integer describing the number of data elements in *set value* in Y dimension.

### 5.3.4 Attribute aliases

A *full attribute name* consists of *device name* and *attribute name* separated by "/" character.

See *RFC 2/Device* for the definition of `<device-name>`.

An Attribute MAY have associated *alias*, a string which can be used in place of *full attribute name* to address the Attribute. An Attribute MUST NOT have more than one *alias*.

An *alias* MUST be stored in the database.

An *alias* MUST be unique across the whole Tango system.

An *alias* MAY contain any character except that it MUST NOT contain any of `/`, (space), `#`, `:`, `->`. It MAY be empty. See `<attribute-alias>` specification below.

### 5.3.5 Attribute events

An Attribute can send following events:

- *PERIODIC*,
- *CHANGE*,
- *ARCHIVE*,
- *DATA READY*,
- *ATTR CONF*,
- *USER*,

Events can be sent automatically by a polling mechanism, automatically by the Tango system or manually from the device server code.

The polling mechanism can send PERIODIC, CHANGE and ARCHIVE events. In order to send events via the polling mechanism, the polling for the Attribute MUST be enabled. See *Attribute properties* section for conditions upon which the events are sent with the polling enabled. See (RFC-10/RequestReply) for more details on polling configuration.

The Tango system MUST send ATTR CONF event whenever Attribute configuration is changed. See *Attribute properties* section for a list of modifiable attribute Properties.

The device server code can manually send *CHANGE*, *ARCHIVE*, *DATA READY* and *USER* events without the need to configure the polling for the Attribute.

### 5.3.6 Attribute naming schema

Formal specification of Attribute name is given below:

```
attribute-name = 1*attribute-name-char
attribute-name-char = %d48-57 / %d65-90 / %d97-122 / "_" ; 0-9 / A-Z / a-z / _

full-attribute-name = device-name "/" attribute-name

attribute-alias = *attribute-alias-char
attribute-alias-char = OCTET ; except %x00 "/" " " "#" ":" "->"
```

CHAPTER

SIX

# 5/PROPERTY

This document describes the Tango Property model.

See also: *1/Tango*, *2/Device*, *4/Attribute*, *6/Database*, *9/DataTypes*

## 6.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 6.2 Tango Property Specification

This specification is intended to formally document the Tango Property model. Runtime representation of Property in conforming Tango implementations MUST follow this specification.

### 6.2.1 Goals

A Property represents a configuration parameter in Tango.

### 6.2.2 Use Cases

Here are some Property use cases:

- To connect a real network equipment a Property can define the target IP address or hostname
- To change the representation of an information in order to be more user friendly a Property can define a new data format for the graphical user interface to convert the raw data
- To configure some alarms on Attribute Value (See RFC-4)
- To store the Attribute value of a Memorized Attribute (See RFC-4)
- To configure the labels associated to the possible values of a DevEnum Attribute
- ...

## 6.3 Specification

A Property is a strict definition of a pair of key/value:

- The Property SHALL have one key, called Property Name
- The Property SHALL have one value, which could be empty, called Property Value

There are 4 kinds of Properties:

- Class Property: a Property related to a Device Class (RFC-2).
- Device Property: a Property related to a Device (RFC-2).
- Attribute Property: a Property related to an Attribute (RFC-4).
- Free Property: a Property defined for the entire Control System (RFC-1) i.e not related to a specific Class, Device or Attribute.

A Property MAY be persisted in the Tango Database (See RFC-1 and RFC-6) or into a file if no Tango Database is used.

Device Properties and Class Properties MAY have a *Default Value* which is used if the Property Value has not been persisted.

A Tango Device Property MAY be defined as *Mandatory in Database*. This metadata can be used when the device server programmer requires this Property to be persisted. A Device Property defined as *Mandatory in Database* SHALL not have a Default Value.

Device Property and Class Property MAY have a Type metadata. It is RECOMMENDED to support the following Types for Device Properties and Class Properties (See RFC-9):

- DevBoolean
- DevShort
- DevUShort
- DevLong
- DevULong
- DevLong64
- DevULong64
- DevFloat

- DevDouble
- DevString
- DevVarShortArray
- DevVarLongArray
- DevVarLong64Array
- DevVarFloatArray
- DevVarDoubleArray
- DevVarStringArray

It is RECOMMENDED to provide a way to retrieve a persisted Property Value and to convert it to the above types.

It is RECOMMENDED to support *Not a Number (NaN)* as well as *-infinity (-inf)* and *+infinity (+inf)* DevFloat and DevDouble values.

### 6.3.1 Naming convention

- The Property's Name SHALL use the following convention:

```
alphanum= ALPHA/DIGIT
underscore = %x5F
device_property_name = 1*1ALPHA 0*254(alphanum / underscore)
class_property_name = 1*1ALPHA 0*254(alphanum / underscore)
free_property_name = 1*1ALPHA 0*254(alphanum / underscore)
attribute_property_name = 1*1(ALPHA / underscore) 0*254(alphanum / underscore)
```

CHAPTER

SEVEN

# 6/DATABASE

This document describes the Tango Database specification.

## 7.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 7.2 Related RFC specifications (TBD)

- X/Device Model
- X/Attributes
- X/Properties

## 7.3 Tango Database specification

This specification is intended to formally document the Tango Database. Runtime implementations of the Database in conforming Tango implementations MUST follow this specification.

### 7.3.1 Goals of the Database

- provide a means of defining and querying devices.
- provide means for device servers to be configured from central control
- serve as nameserver such that the Devices are accessible by name

### 7.3.2 Features of the Database

- store configuration data (properties) used at startup of a device server
- act as nameserver by storing the dynamic network addresses
- act as permanent store of dynamic settings which need to be memorized
- ensure the uniqueness of device name and of aliases
- keep list of controlled servers which are to be started and run on a particular host
- limited set functionality provide by file as alternative to network database

### 7.3.3 Use Cases of the Database

- register a new device server
- stores the last user setting point of the current provided by a power supply
- define persistent configuration properties to be used by a camera every time it starts up
- allow a subscriber or client to get connect to a stepper motor
- define properties for all instances of a class of vaccuum pump
- provide the network ID of a device such as an electrometer, so a user can check if it is online or not
- a user would like to rollback the configuration of a Device when its initialization is failing
- remove “empty” Devices from Database to clean it up
- get all Devices of a given class of temperature gauges
- user wants to see the history (last ten values) of a Property
- a monitor tool wants the Info of a not running detector Device (exported, host, server)
- a boot-up process wants to get Starter Level configuration for a list of Devices

## 7.4 Specification

The database SHALL provide a means of defining and querying devices.

As the Tango database service is a central service of the Tango control system, it SHALL be designed to be highly available and support a high level of client requests (e.g we have peaks of 3500 SQL requests/ s at SOLEIL on the accelerators control system).

The database service SHALL be implemented as a Tango Device Server.

The database service SHALL be implemented with:

- a frontend: as Tango Device Server

- a backend: a query and persistency engine (i.e. SQL or NoSQL engine), or a lighter ( i.e. reduced set of functionalities) engine with a single file per device server.

The Database Device SHALL have no Class Properties or Device Properties.

In the Tango Control System, its database service SHALL be available on a pre-known network address.

Clients and subscribers SHOULD access the database service via TANGO commands requested on the database device.

As a Tango Device Server, the database SHALL provide State and Status.

A limited set of the database functionality to support Tango Device Server configuration SHALL be available as a read-only file. This is an option for use in the absence of a network service.

The database SHALL enforce uniqueness of device names and aliases.

The Database SHALL enforce the naming convention defined in all Tango RFCs for device name, the command name, the attribute name, the property name, the device alias name and the device server name.

The device name, its domain, member and family fields and its alias maximum sizes SHALL BE at least:

- device name: 255
- domain field: 85
- family field: 85
- member field: 85
- device alias name: 255

## 7.5 List of Commands Grouped by Functionality

### 7.5.1 Devices

The database SHALL implement the following device-related commands:

- DBAddDeviceDbDeleteDevice
- DbGetClassForDevice
- DbGetClassInheritanceForDevice
- DbGetDeviceClassList
- DbGetDeviceDomainList
- DbGetDeviceFamilyList
- DbGetDeviceInfo
- DbGetDeviceList
- DbGetDeviceWideList
- DbGetDeviceMemberList
- DbGetDeviceServerClassList

### 7.5.2 Classes

The database SHALL provide a means of querying device server classes.

The database SHALL implement the following class-related commands:

- DbGetClassList
- DbGetDeviceClassList
- DbGetDeviceServerClassList

### 7.5.3 Properties

The database SHALL provide a means of setting and getting properties for classes, devices, attributes, pipes and free objects.

The database SHALL provide a means to retrieve the history of property value settings.

The database SHALL implement the following property-related commands:

#### ClassAttributeProperty

- DbDeleteClassAttributeProperty
- DbGetClassAttributeProperty
- DbGetClassAttributeProperty2
- DbGetClassAttributePropertyHist
- DbPutClassAttributeProperty
- DbPutClassAttributeProperty2

#### ClassProperty

- DbDeleteClassProperty
- DbGetClassProperty
- DbGetClassPropertyHist
- DbGetClassPropertyList
- DbPutClassProperty

#### DeviceAttributeProperty

- DbDeleteDeviceAttributeProperty
- DbGetDeviceAttributeProperty
- DbGetDeviceAttributeProperty2
- DbGetDeviceAttributePropertyHist
- DbPutDeviceAttributeProperty
- DbPutDeviceAttributeProperty2

- DbDeleteAllDeviceAttributeProperty

#### DeviceProperty

- DbDeleteDeviceProperty
- DbGetDeviceProperty
- DbGetDevicePropertyHist
- DbGetDevicePropertyList
- DbPutDeviceProperty

#### Free Object Property

- DbDeleteProperty
- DbGetProperty
- DbGetPropertyHist
- DbGetPropertyList
- DbPutProperty

#### ClassPipeProperty

- DbGetClassPipeProperty
- DbDeleteClassPipeProperty
- DbPutClassPipeProperty
- DbGetClassPipePropertyHist

#### DevicePipeProperty

- DbGetDevicePipeProperty
- DbDeleteDevicePipeProperty
- DbDeleteAllDevicePipeProperty
- DbPutDevicePipeProperty
- DbGetDevicePipePropertyHist

### 7.5.4 Attributes

The database SHALL provide for the persistent storage of Attributes of classes and devices.

The database SHALL implement the following attribute-related commands:

#### ClassAttribute

- DbDeleteClassAttribute
- DbGetClassAttributeList

#### DeviceAttribute

- DbDeleteDeviceAttribute
- DbGetDeviceAttributeList

### 7.5.5 Aliases

The database SHALL provide a means of setting and getting a mapping from alias to device name.

The database SHALL provide a means of setting and getting a mapping from alias to device attribute name.

The database SHALL implement the following attribute-related commands:

#### AttributeAlias

- DbDeleteAttributeAlias
- DbGetAttributeAlias
- DbGetAttributeAliasList
- DbPutAttributeAlias
- DbGetAttributeAlias2
- DbGetAliasAttribute

#### DeviceAlias

- DbDeleteDeviceAlias
- DbGetAliasDevice
- DbGetDeviceAlias
- DbGetDeviceAliasList
- DbPutDeviceAlias

### 7.5.6 Hosts/Servers

The database SHALL provide a means of defining hosts and servers.

The database SHALL implement the following commands:

#### Host

- DbGetHostList
- DbGetHostServerList
- DbGetHostServersInfo

#### Server

- DBAddServer
- DbDeleteServer
- DbGetServerList
- DbGetServerNameList
- DbRenameServer

#### ServerInfo

- DbDeleteServerInfo
- DbGetServerInfo
- DbPutServerInfo

### 7.5.7 Pipes

The database SHALL provide a means of managing information about pipes.

The database SHALL implement the following pipe-related commands:

#### ClassPipe

- DbDeleteClassPipe
- DbGetClassPipeList

#### DeleteDevicePipe

- DbGetDevicePipeList

### 7.5.8 Free Objects

The database SHALL provide a means of querying Free Objects.

The database SHALL implement this free object-related command:

- DbGetObjectList

### 7.5.9 Exported Devices

The database SHALL provide a means to register the network address of a running Device Server and events.

The database SHALL implement the following free registry-related commands:

- DbExportDevice
- DbUnExportDevice
- DbGetDeviceExportedList
- DbGetExportedDeviceListForClass
- DbImportDevice
- DbExportEvent
- DbImportEvent
- DbUnExportEvent
- DbUnExportServer

### 7.5.10 Miscellaneous

The database SHALL provide the following miscellaneous commands.

- DbGetInstanceNameList
- DbGetForwardedAttributeListForDevice
- DbInfo
- ResetTimingValues
- DbGetDataForServerCache
- DbMySqlSelect
- DbGetCSDbServerList

## 7.6 List of Database commands and description

## 7.7 Database as a file

For Device servers not able to access the Tango database (most of the time due to network route or security reason), it is possible to start them using a file instead of a real database. This is done via the Device server command line option:`-file=<file name>`

In this case, the Database SHALL handle the following functionality using the specified file:

- getting, setting and deleting class properties
- getting, setting and deleting device properties
- getting, setting and deleting class attribute properties
- getting, setting and deleting device attribute properties

The following set of limitations exist:

- no check that the same Device server is running twice
- no Device or attribute alias name
- in case of several Device servers running on the same host, the user must manually manage a list of already used network ports

### 7.7.1 File syntax

The file is an ASCII file. It syntax is defined completely in .

Generally, comment starts with the '#' character and a blank line is ignored.

#### File syntax: Devices definition.

DEVICE is the keyword to declare a device(s) definition sequence. The general syntax SHALL be`<DS name>/<inst name>/DEVICE/<Class name>: dev1,dev2,dev3` Device(s) name can follow on next line if the last line character is '\'

#### File syntax: Device property definition.

The general device property SHALL be`<device name>-><property name>: <property value>`In case of array, the array element delimiter is the character ','. Array definition can be splitted on several lines if the last line character is '\'. Allowed characters after the ':' delimiter are space, tabulation or nothing.

A device string property with special characters (spaces). The '' character is used to delimit the string

#### File syntax: Device attribute property definition.

The general device attribute property syntax SHALL be`<device name>/<attribute name>-><property name>: <property value>`Allowed characters after the ':' delimiter are space, tabulation or nothing.

#### File syntax: Class property definition.

The general class property syntax SHALL be`CLASS/<class name>-><property name>: <property value>`

CLASS is the keyword to declare a class property definition. Allowed characters after the ':' delimiter are space, tabulation or nothing. The '' characters around the property value are mandatory due to the '/' character contains in the property value.

## 7.8 Related documentation

Tango Controls Documentation

PyTango Documentation Release

DataBase Tango Cpp Class

Certified Database Device Server Reference Implementation

The TANGO device Server model

Multiple database servers within a Tango control system

Memorized attribute

Introduction to Astor

Starter Tango Cpp Class

Using Database Object

Device server using file as database

Database API

The property file syntax

CHAPTER

EIGHT

# 7/THE TANGO PIPE SPECIFICATION

This specification describes the Tango Pipe feature of a Device.

## 8.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 8.2 Goals

Tango Pipe is an Attribute [RFC-4] with custom data structure - the pipe name and pipe description. But in contrary of Commands or Attributes, a pipe does not have a pre-defined data type: it may be of any basic Tango data type (or array of) and this may change every time a pipe is read or written.

## 8.3 Use Case

1. To pass big pieces of data structure at a time
2. To define complex data structures to act as a single Attribute

## 8.4 Specification

Device MAY define zero, one or more Pipes.

Pipe MUST have static metadata:

- name
- description
- label
- level
- writeType
- writable

```
name = TangoName

description = DevString [link to RFC-9 Data Types]

label = DevString [link to RFC-9 Data Types]

level = "OPERATOR"/"USER"

writeType = "PIPE_READ"/"PIPE_WRITE"

writable = DevBoolean
```

Pipe name MUST follow Tango naming conventions.

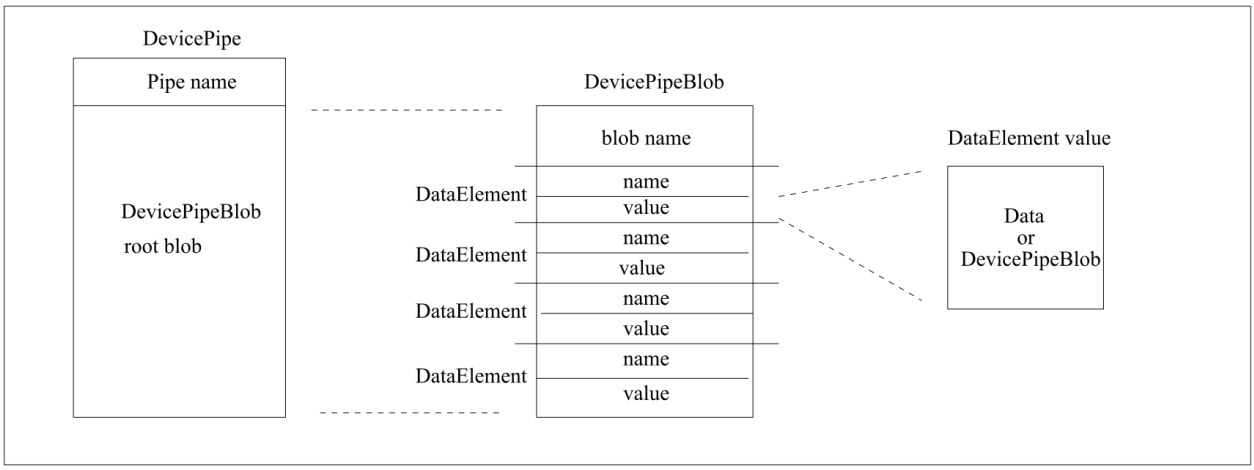
Pipe with writeType=”PIPE_READ” MUST be read only.

Pipe data structure MUST be:

- Key(s) -> Value(s)

```
Key = DevString [link to RFC-9 Data Types]

Value = Scalar or Array of Any Tango compliant type or Pipe [link to RFC-9 Data Types]
```

In Tango 9 “DevPipeBlob” is used to describe Value (see image below).




**V10 NOTE:** Attribute being a corner case of a Pipe can be considered as a derivative of a Pipe.

### 8.4.1 Pipe events

Pipe MUST NOT be polled.

Pipe MAY emit an event via Server API call.

### 8.4.2 Server/Client APIs

Server API MUST provide a set of methods for a Server Client code to define a Pipe.

Server API MUST provide a way to fill the Pipe with data.

Client API MAY provide a way to obtain Pipe schema.

Client API MUST provide methods for a client to fetch Pipe data from a Device.

Client API MUST provide methods to traverse Pipe structure.

CHAPTER

NINE

# 8/DEVICE SERVER MODEL

This document provides the specification of the Device Server.

See also: *2/Device*

## 9.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 9.2 Tango Device Server specification

This specification formally document the Tango Device Server model. Runtime implementation of the Device Server model in conforming Tango implementations MUST follow this specification.

### 9.2.1 Goals

In a Tango Control System a Device Server is in charge of managing the life cycle and the communication protocol of one or several Tango Devices. The Device Server can also offer different services to the clients as well as for the Device.

Additionally, it aims to:

- Provide a blackbox that registers the operations executed on every Device
- Be usable as a Device

### 9.2.2 Use Cases

The main use cases for Device Server are:

- to manage several Devices in the same process.
- to share the same channel of communication
- to provide the connection protocols for accessing the managed Device
- to provide different services like logging
- to serialise the operations per Device
- to sign up or sign out to the central registry

## 9.3 Specification

The Device Server manages the communication with one or several Devices. Additionally it enforces the compatibility of behaviour with the Tango Control System and offers different meta services.

The Device Server SHALL implement the following specifications to control and monitor the Device:

- A Device Server represents the process which manages one or several Devices
- A Device Server SHALL manage at least one Device.
- A Device Server MAY manage Devices belonging to different Classes
- A Device MUST NOT be managed by more than one Device Server
- A Device Server instance reference SHALL be unique in a Tango Control system
- The communication to the Device Server SHALL be done through a Tango Device so called Administration Device or Admin Device .

The specification introduces as well the definition of the processes which are used for the Administration Device Name:

- A Server is a type of program which execute a Device Server
- An Instance is an unique identifier as several processes of the same Server MAY exist in a Tango Control System
- A Server Instance is finally what identify the actual process in the Tango Control System
- A Server Instance CAN only execute one Device Server

The name of the Administration Device SHALL follow this convention:

```
administration-device-name = dserver "/" server "/" instance
dserver = "dserver" ; dserver is a reserved domain following the device-name␣
↪convention.
server = family
instance = member
```

- The family part of the Administration Device Name SHALL be the Server name
- The member part of the Administration Device Name SHALL be the Instance

### 9.3.1 Device Server interface

| Property Name | description |
|---|---|
| polling_threads_pool_conf | TODO |

| Command Name | Argument input | Argument Output | description |
|---|---|---|---|
| AddLoggingTarget | DevVarStringArray | DevVoid | Define the level of Logging(*RFC-14* |
| AddObjPolling | DevVarLongStringArray | DevVoid | Polling related command (RFC-10) |
| DevLockStatus | DevString | DevVarLongStringArray | Request Replay command (RFC-10)) |
| DevPollStatus | DevString | DevVarStringArray | Polling related command (RFC-10)) |

| DevRestart | DevString | DevVoid | | EventConfirmSubscription | DevVarStringArray | DevVoid | | EventSubscriptionChange | DevVarStringArray | DevLong | | GetLoggingLevel | DevVarStringArray | DevVarLongStringArray | | GetLoggingTarget | DevString | DevVarStringArray | | Init | DevVoid | DevVoid | | Kill | DevVoid | DevVoid | | LockDevice | DevVarLongStringArray | DevVoid | | PolledDevice | DevVoid | DevVarStringArray | | QueryClass | DevVoid | DevVarStringArray | | QueryDevice | DevVoid | DevVarStringArray | | QuerySubDevice | DevVoid | DevVarStringArray | | QueryWizardClassProperty | DevString | DevVarStringArray | | QueryWizardDevProperty | DevString | DevVarStringArray | | ReLockDevices | DevVarStringArray | DevVoid | | RemObjPolling | DevVarStringArray | DevVoid | | RemoveLoggingTarget | DevVarStringArray | DevVoid | | RestartServer | DevVoid | DevVoid | | SetLoggingLevel | DevVarLongStringArray | DevVoid | | StartLogging | DevVoid | DevVoid | | StartPolling | DevVoid | DevVoid | | State | DevVoid | DevState | | Status | DevVoid | DevString | | StopLogging | DevVoid | DevVoid | | StopPolling | DevVoid | DevVoid | | UnLockDevice | DevVarLongStringArray | DevLong | | UpdObjPollingPeriod | DevVarLongStringArray | DevVoid | | ZmqEventSubscriptionChange | DevVarStringArray | DevVarLongStringArray |

### 9.3.2 The Device Export sequence

The phase called Device Export is intended to prepare all the managed Device for operation and accessibility in the Tango Control System by the Device Server:

- The Device Server MUST take in charge of the Device Export
- The Device Export SHOULD bring the Device at the same State than after an Init Command.
- The Device Export SHOULD execute the Device Sign up of the Database(*RFC-6*), in order to be accessible locally and via network.

### 9.3.3 The Device Unexport sequence

The phase called Device Unexport is intended to prepare the all Device for stopping operate.

- The Device Server MUST take in charge of the Device Unexport
- The Device Unexport SHALL gracefully stop the Device operation, even if there are requests in progress.
- The Device Unexport SHALL gracefully stop the communication with any Tango Client
- The Device Unexport SHOULD unregister from the Tango Control system by executing the Device Sign Out of the Database(*RFC-6*), in order to notify that all managed Device are no longer accessible

CHAPTER

TEN

# 9/DATA TYPES

This document describes Tango Controls Data Types specification version 5.0.

See also: *1/TANGO*, *3/Command*, *4/Attribute*, *5/Property*, *6/Database*, *7/Pipe*,

## 10.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 10.2 Tango Data Types Specification

Data Types are specifying format and limits of pieces of information exchanged between Tango Controls components.

### 10.2.1 Goals

Data Types specification aims to standardise a way how different kind of information is entered, displayed and processed within a Tango Controls system irrespectively to its encoding in a transport protocol or computers' CPU architecture. It also allows to distinguish different kinds of numeric data, string data, enumerations or multi-values data.

### 10.2.2 Use Cases

There are many use cases for Data Types. Some of them are listed below:

- To specify expected data format for attributes, commands and pipes.
- To display in the same way a numeric value read from a device when some of them are using big-endian encoding and others use little-endian encoding.
- To limit information processed to a practical level by choosing a Data Type of certain precision. For example, a value of Data Type DevShort typically provides less precise information and consumes less computing resources than a value of Data Type DevLong.

### 10.2.3 DataType

- Any value of Properties, Attributes, Pipes' fields and Commands' input and output arguments SHALL have defined its DataType.
- DataType SHALL be as follows:

```
DataType = DevVoid

DataType =/ DevBoolean

DataType =/ DevShort / DevLong / DevLong64
DataType =/ DevUChar / DevUShort / DevULong / DevULong64

DataType =/ DevFloat / DevDouble

DataType =/ DevString

DataType =/ DevVarBooleanArray / DevVarDoubleArray / DevVarFloatArray
DataType =/ DevVarShortArray / DevVarLongArray / DevVarLong64Array / DevVarCharArray
DataType =/ DevVarStringArray
DataType =/ DevVarUShortArray / DevVarULongArray / DevVarULong64Array

DataType =/ DevEncoded / DevVarEncodedArray

DataType =/ DevVarLongStringArray / DevVarDoubleStringArray

DataType =/ DevState / DevVarStateArray

DataType =/ DevEnum

DataType =/ DevPipeBlob

DataTpe =/ DevFailed
```

### 10.2.4 Numeric

#### DevBoolean

DevBoolean is a data type which values represents logical state of true or false.

- DevBoolean SHALL use the following name:

```
DevBoolean = "DevBoolean"
```

- Values which DataType is DevBoolean SHALL carry one bit of information.

#### Integer data types

DevShort, DevLong, DevLong64 represent signed integer values. DevUChar, DevUShort, DevULong and DevULong64 represent unsigned values.

- Integer DataType SHALL have the following names:

```
DevShort = "DevShort"
DevLong = "DevLong"
DevLong64 = "DevLong64"
DevUChar = "DevUChar"
DevUShort = "DevUShort"
DevULong = "DevULong"
DevULong64 = "DevULong64"
```

- Values of integer data types SHALL be in defined ranges and the information they carry SHALL have number of bits as specified in the table below:

| DataType | Range | Bits of information |
|---|---|---|
| DevShort | -(2^15) to (2^15)-1 | 16 |
| DevLong | -(2^31) to (2^31)-1 | 32 |
| DevLong64 | -(2^63) to (2^63)-1 | 64 |
| DevUChar | 0 to 255 | 8 |
| DevUShort | 0 to (2^16)-1 | 16 |
| DevULong | 0 to (2^32)-1 | 32 |
| DevULong64 | 0 to (2^64)-1 | 64 |

#### Floating point data types

DevFloat and DevDouble represent floating point values.

- DevFloat and DevDouble SHALL use the following names:

```
DevFloat = "DevFloat"
DevDouble = "DevDouble"
```

- Values of DevFloat type SHALL be of IEEE 754 single precision floating-point format and MAY be encoded in 32 bits.
- Values of DevDouble type SHALL be of IEEE 754 double precision floating-point format and MAY be encoded in 64 bits.

### String data type

DevString values represent strings (sequence of characters).

- DevString SHALL use the following names:

```
DevString = "DevString"
```

- Values of DevString SHALL be according to the string-value rule defined as follows:

```
string-value = *CHAR
```

### Sequence data types

DevVarBooleanArray, DevVarShortArray, DevVarLongArray, DevVarLong64Array, DevVarCharArray, DevVarUShortArray, DevVarULongArray, DevVarULong64Array, DevVarFloatArray, DevVarDoubleArray, DevVarStringArray, DevVarEncodedArray, DevVarStateArray are Sequence DataTypes. Sequence data types are sequences (arrays) of values (elements) of matching DataType, which are respectively, DevBoolean, DevShort, DevLong, DevLong64, DevUChar, DevUShort, DevULong, DevULong64, DevFloat, DevDouble, DevString, DevEncoded, DevState.

- Names of the sequence data types SHALL be as follows:

```
DevVarBooleanArray = "DevVarBooleanArray"
DevVarShortArray = "DevVarShortArray"
DevVarLongArray = "DevVarLongArray"
DevVarLong64Array = "DevVarLong64Array"
DevVarCharArray = "DevVarCharArray"
DevVarUShortArray = "DevVarUShortArray"
DevVarULongArray = "DevVarULongArray"
DevVarULong64Array = "DevVarULong64Array"
DevVarFloatArray = "DevVarFloatArray"
DevVarDoubleArray = "DevVarDoubleArray"
DevVarStringArray = "DevVarStringArray"
DevVarEncodedArray = "DevVarEncodedArray"
DevVarStateArray = "DevVarStateArray"
```

- Value of sequence DataType SHALL follow rule:

```
sequence-value = *element
```

Where <element> follows specification of value of the matching <DataType>

- Sequence DataType SHALL provide an interface to determine number of its elements.
- Sequence DataType SHALL provide an interface to index its elements.

### Structures

Value of the DevEncoded type is a structure to carry binary data with application and context specific meaning.

- Name of DevEncoded type SHALL be as follows:

```
DevEncoded = "DevEncoded"
```

- Value of <DevEncoded> SHALL be according to the <encoded-value> rule:

```
encoded-value = encoded_format encoded_data
                ; elements order implied by the above rule MAY be ignored
```

;where <encoded_format> has type DevString and <encoded_data> has type DevVarCharArray.

DevVarLongStringArray and DevVarDoubleStringArray values are structures to carry sequences of numbers (integer or double precision floating point, respectively) along with sequences of text/string data.

- Names DevVarLongStringArray and DevVarDoubleStringArray SHALL follow the following rules:

```
DevVarLongStringArray = "DevVarLongStringArray"
DevVarDoubleStringArray = "DevVarDoubleStringArray"
```

- Values of DevVarLongStringArray and DevVarDoubleStringArray SHALL follow rules <long-string-value> and <double-string-value>, respectively:

```
long-string-value = lvalue svalue
double-string-value = dvalue svalue
                      ; elements order implied by the above rules MAY be ignored
```

;where <lvalue> has type DevVarLongArray, <dvalue> has type DevVarDoubleArray and <svalue> has type DevVarStringArray.

### State

DevState type standardizes a way of describing state of Device. Its values are from predefined set of 14 names.

- DevState name SHALL be according to the following rule:

```
DevState = "DevState"
```

- DevState type SHALL be an enumeration type with labels following the rule <dev-state-label>:

```
dev-state-label = "ALARM" / "INSERT" / "STANDBY" / "CLOSE" / "MOVING" / "UNKNOWN" /
↪"DISABLE" / "OFF"
dev-state-label =/ "EXTRACT" / "ON" / "FAULT" / "OPEN" / "INIT" / "RUNNING"
```

### Enumeration

DevEnum data type allows to assign string Labels to DevShort values. It is valid only for Attributes.

- DevEnum name SHALL follow the following rule:

```
DevEnum = "DevEnum"
```

- Label values of DevEnum SHALL follow the rule <devenum-label>:

```
devenum-label = 1*VCHAR
```

- DevEnum labels SHALL be unique in the context of an Attribute.
- Each of DevEnum labels SHALL correspond to one and unique DevShort value.
- Related DevShort values SHALL be consecutive and SHALL start with 0.
- Values of DevEnum DataType SHALL be transported as values of DevShort.
- For any Attribute of DevEnum data type, list of Labels SHALL be available as `enum_labels` Attribute property (see *4/Attribute*).
- Tango Controls SHALL allow a Tango Client to retrieve labels associated with the DevShort value.

### Pipe

DevPipeBlob is a Data Type to transfer data related to Pipes (see 7/Pipe).

- DevPipeBlob SHALL have the following name:

```
DevPipeBlob = "DevPipeBlob"
```

- A value of DevPipeBlob type SHALL follow the <pipe-blob-value> rule:

```
pipe-blob-value = name blob-data
                  ; elements order implied by the above rule MAY be ignored

name = 1*VCHAR

blob-data = *blob-data-element

blob-data-element = name (value / blob-data)
                    ; elements order implied by the above rule MAY be ignored
```

Where <value> SHALL be a value of one of the following data types: DevBoolean, DevShort, DevLong, DevLong64, DevFloat, DevDouble, DevChar, DevUShort, DevULong, DevULong64, DevString, DevState, DevState, DevEncoded, DevVarBooleanArray, DevVarShortArray, DevVarLongArray, DevVarLong64Array, DevVarFloatArray, DevVarDoubleArray, DevVarCharArray, DevVarUShortArray, DevVarULongArray, DevVarULong64Array, DevVarStringArray, DevVarStateArray, DevState, DevVarEncodedArray.

### Exceptions

Results of failed operations (exceptions) within the Tango Controls are sent as values/messages of DevFailed or MultiDevFailed data type.

- DevFailed type name is as follows:

```
DevFailed = "DevFailed"
```

- Any value of DevFailed type SHALL follow <dev-failed-value> rule:

```
dev-failed-value = 1*error

error = reason severity desc origin

reason = *VCHAR ; SHOULD be a symbolic name, without whitespace, which points the␣
↪reason for the failure and is standardized for some errors thrown by the API.␣
↪Examples: "API_AttrOptProp", "API_AttrNotPolled","API_CmdNotPolled", "API_
↪CommandNotFound"
severity = "WARN" | "ERR" | "PANIC"
desc = *VCHAR ; SHOULD describe the failure in more details
origin = *VCHAR ; SHALL identify the operation or the tango object which caused the␣
↪failure, eg. function name
```

- MultiDevFailed type name is as follows:

```
MultiDevFailed = "MultiDevFailed"
```

- Any value of MultiDevFailed type SHALL follow <multi-dev-failed-value> rule:

```
multi-dev-failed-value = 1*named-dev-error

named-dev-error = name index-in-call 1*error

name = *VCHAR
index-in-call = num-val ; of long data type (32 bits) in the range of 0 to (2^31)-1

error = reason severity desc origin

reason = *VCHAR ; SHOULD be a symbolic name, without whitespace, which points the␣
↪reason for the failure and is standardized for some errors thrown by the API.␣
↪Examples: "API_AttrOptProp", "API_AttrNotPolled","API_CmdNotPolled", "API_
↪CommandNotFound"
severity = "WARN" | "ERR" | "PANIC"
desc = *VCHAR ; SHOULD describe the failure in more details
origin = *VCHAR ; SHALL identify the operation or the tango object which caused the␣
↪failure, eg. function name
```

CHAPTER

# ELEVEN

# 12/PUBLISHER-SUBSCRIBER PROTOCOL

## 11.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 11.2 Publisher-Subscriber protocol Specification

It is a relationship between Tango client (Client) and Tango server (Server). Based on criteria negotiated as part of a subscription, Tango client will receive updates. Beyond a set of basic requirements, various refinements are addressed. These refinements include: periodicity of updates, filtering out updates based on Tango server configuration and fault tolerance.

## 11.3 Basic Concepts

This documents refers to other Tango Controls RFCs, corresponding link is always given in this case. Here is given a brief description of the main concepts. This document targets arbitrary software developer(s) who are to implement *Tango Client Library* and *Tango Server Library*.

*Tango Server* (Server) - a process that runs on a machine.

*Tango Device* (Device) - for more details see RFC-2

*Tango Admin Device* (Admin) - for more details see [RFC-8]

*Tango Client* (Client) - a process that instantiates communication with *Tango Server* using Request-Reply protocol [RFC-10].

*Tango Client Library* (client kernel library) - library that encapsulates communication with *Tango Server* and provides *Tango Client API* to a user.

*Tango Client API* (client API) - an API that user uses to develop applications based on communication with *Tango Server*s.

*Tango Server Library* (server kernel library) - library that encapsulates *Tango Server* features and provides *Tango Server API* to a user.

*Tango Server API* (server API) - an API that user uses to develop *Tango Device* functionality.

*Upstream Tango Server/Device* (upstream server/device) - a *Tango Device* to which updates *client* subscribes for.

*Polling thread* (polling thread) - a thread within *Tango Server* that polls periodically defined *Tango Device*'s attributes/sometimes commands.

*Tango Publish-Subscriber protocol* (pub-sub) - is a sub-set of *Tango Request-Reply protocol* [RFC-10] that defines a relationship between *client* and *server* established via a negotiation phase and lasts till explicitly broken. During this relationship client receives updates published by *upstream server*. Updates are sourced from server's *polling thread* or directly through *server API*. Updates are filtered according to *device*'s configuration.

*Tango Event* (event) - an update sent by *server*.

*Tango Event Callback* (callback) - a piece of code that *client library* executes when an update is received.

*Channel* - virtual client-server point-to-point connection for transmitting events

## 11.4 Definitions

This section offers a number of ABNF formal definitions for entities repeatedly used below in this document.

> **NOTE**: in the below ABNF some basic definitions are used as rules, spaces are replaced with '_'
>
> **NOTE**: in the below ABNF data types sometimes embeded into the rule e.g. bool:isExcept is equivalent to isExcept = true / false

```
endpoint = URL ; usually with port

SUBSCRIBER_INFO = upstream_device attribute_name action EVENT_TYPE client_idl_ver

SUBSCRIPTION_INFO = server_library_release_ver upstream_device_idl_ver zmq_sub_event_
↪hwm rate ivl zmq_release_ver heartbeat_endpoint event_endpoint 1*alternate_
↪heartbeat_endpoint 1*alternate_event_endpoint HEARTBEAT_CHANNEL EVENT_CHANNEL

EVENT_INFO = EVENT_DATA/1*EVENT_ERROR bool:isExcept  ; see below

EVENT_DATA = ATT_CONF_DATA/PIPE_DATA/DATA_READY_DATA/INTERFACE_CHANGE_DATA/ATTRIBUTE_
↪DATA

ATT_CONF_DATA = ATTRIBUTE_PROPERTIES ; Attribute properties from [RFC-4](https://
↪github.com/tango-controls/rfc/blob/draft/4/Attribute.md#attribute-properties)

PIPE_DATA =  int:size name timeVal PIPE_BLOB ; pipeBlob must be defined in [RFC-
↪7](https://github.com/tango-controls/rfc/blob/draft/7/Pipe.md)

DATA_READY_DATA = attribute_name data_type int:counter

INTERFACE_CHANGE_DATA = *ATTRIBUTE_PROPERTIES *COMMAND_META_INFO bool:deviceStarted

ATTRIBUTE_DATA = ATTRIBUTE_DEFINITION ; Attribute definition from [RFC-4](https://
↪github.com/tango-controls/rfc/blob/draft/4/Attribute.md#attribute-definition)
```

(continues on next page)

(continued from previous page)

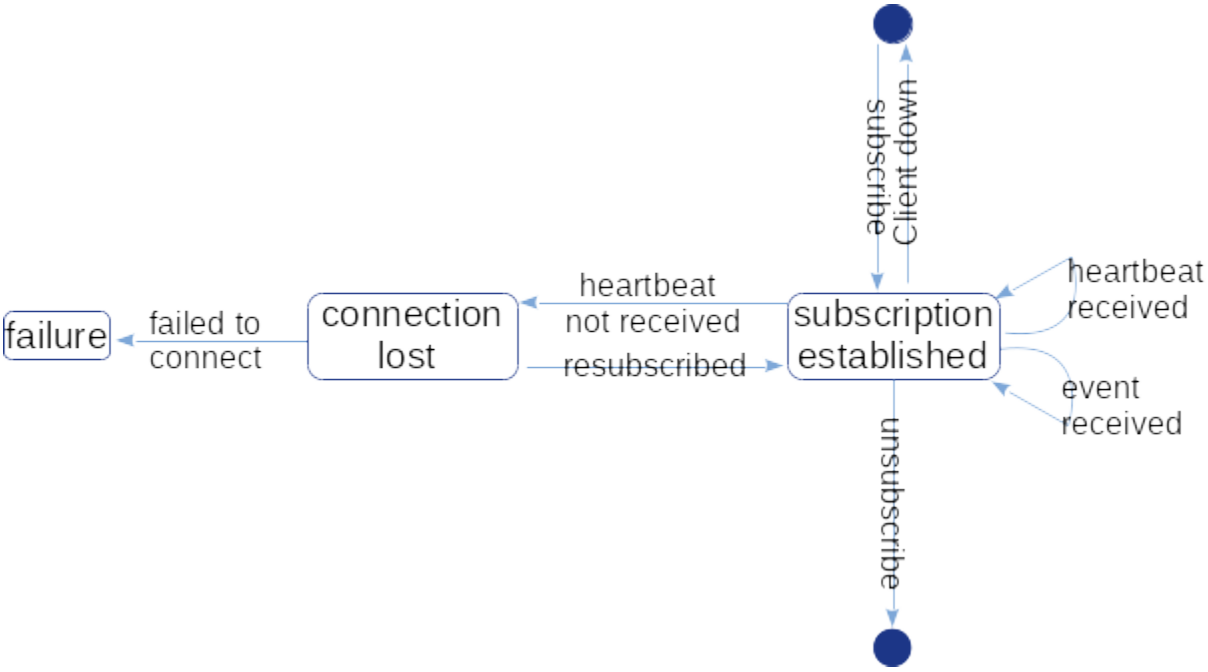
```
EVENT_ERROR = reason severity desc origin

EVENT_TYPE = "INTERFACE_CHANGE"/"PIPE_EVENT"/"ATTR_CONF_EVENT"/"CHANGE_EVENT"/
↪"PERIODIC_EVENT"/"ARCHIVE_EVENT"/"USER_EVENT"

EVENT_CHANNEL = channel

HEARTBEAT_CHANNEL = channel

channel = string ; implementation specific
```

**NOTE**: In Tango V9 EVENT_DATA MAY include source idl version, event type

## 11.5 Runtime requirements

Client and server are up and running. Server is reachable from client i.e. may communicate using Request-Reply protocol [RFC-10].

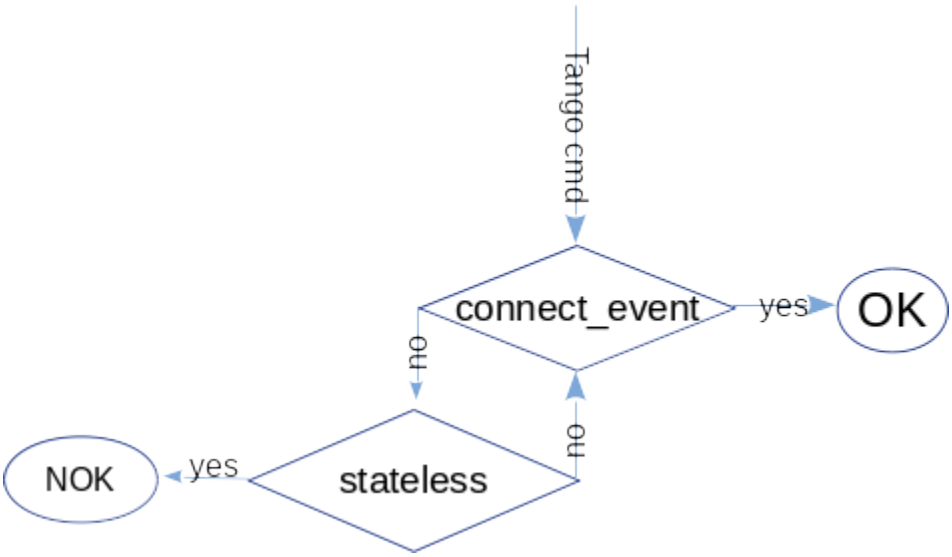
## 11.6 Publisher-Subscriber protocol




The main goal of this protocol implementation is to allow a Client receive events from a Server:

- Server MUST publish events of EVENT_TYPE
- Server library MUST provide an API to publish events of EVENT_TYPE
- Client SHOULD process events
- Client library SHOULD provide user an API to register user defined event callbacks

### 11.6.1 Negotiation




Negotiation phase implies that client and server exchange required data using Tango commands RFC-3, [RFC-13]:

- Client MUST notify server providing SUBSCRIBER_INFO to establish Publisher-Subscriber relationship

  in Tango 9 the above is done by executing EventSubscriptionChange admin command

  Tango 9 forces client to explicitly execute ZmqEventSubscriptionChange to use ZMQ based implementation [RFC-13]

- Server MUST respond to the client providing SUBSCRIPTION_INFO
- Client MUST connect to upstream server using provided SUBSCRIPTION_INFO
- Client SHOULD postpone event subscription process in case of connection failure and try again later

Tango Event System uses concept of channels for delivering events. Channel is established between client and admin. There are two channels per subscription: heartbeat and event.

- Client MUST establish channel or multicast connection to heartbeat/event endpoints
- Server MAY send and client MAY iterate over alternative heartbeat/event endpoints to establish connection
- Client MAY NOT use provided heartbeat/event channels to identify event source

### 11.6.2 Subscription

- Server MUST send events via corresponding EVENT_CHANNEL
- Server MUST send HEARTBEAT via corresponding HEARTBEAT_CHANNEL event every 9 seconds
- Client SHOULD indicate its interest in subscription not later than every 600s

### 11.6.3 Cancel event subscription

- Client MAY cancel its subscription
- Server SHOULD NOT send events to client whose subscription is canceled

### 11.6.4 Fault tolerance

Tango Pub-Sub protocol implies a number of fault tolerance techniques:

- server heartbeat
- transparent client reconnection
- client subscription confirmation
- events client/server buffer
- Client SHOULD try to re-subscribe once HEARTBEAT event is missing
- Client MAY try to re-subscribe indefinitely
- Server MAY cancel client's subscription if it does not confirm its interest
- Client/Server SHOULD buffer the (in/out)coming events to prevent overload and overflow
- Clients SHOULD be notified in case of events overflow
- Client SHOULD be notified in case of missing events

  e.g.: In the current C++ implementation at least, the client callback is executed with an API_MissedEvents error event when the client library detected that some events got lost. There is a mechanism with some counters associated with each event to help detecting this kind of problems

CHAPTER

TWELVE

# 14/THE TANGO LOGGING SERVICE

## 12.1 Introduction

This document describes the Tango Logging Service (TLS). The TLS allows Tango Devices to print messages to be:

- displayed on a console (the classical way)
- sent to a file
- sent to specific Tango device called log consumer
- or more than one of the above at the same time

See also: *2/Device*, *4/Attribute*

## 12.2 Preamble



## 12.3 Goals

This specification will describe the Tango Logging Service (TLS).

It will describe and specify such things as:

- how logging is statically configured for a Device, that is, configured from Device Properties for start up.
- some logging terms: level, target, rft and path.
- what are logging targets: console, file, device
- what is the logging environment variable TANGO_LOG_PATH
- logging levels: OFF, FATAL, ERROR, WARNING, INFO, DEBUG.

- logging commands available to all Devices
- the standard Tango Log Consumer Device and its interface (static mode, dynamic mode).

## 12.4 Use Cases

The Tango Logging Service (TLS) provides the programmer with a convenient way to print information which helps to:

- debug the software
- report on errors
- give regular information to user
- provide timely information of the activities of the Device
- report the version of the softare
- provide clues to the internal changes happening in the Device

At run-time, the TLS supports such activities as:

- changing logging verbosity and destination of any Tango Device
- capture logs to files
- display log entries on the command line console
- interactively view log entries as they are produced by a remote Tango Device
- review logs already written to files
- make graphical dashboards with such tools as Kibana and Grafana

## 12.5 Specification

### 12.5.1 Logging Properties

TLS SHALL use four Device Properties to control Device Server logging at startup. The Properties may be set during runtime in the normal way.

#### logging_level property

SHALL controls the initial logging level of a device. It SHALL have the set of possible values: OFF, FATAL, ERROR, WARN, INFO or DEBUG. This property SHALL be overwritten by the verbose command line option (-v).

#### logging_target property

SHALL give the name of one or more Logging Target (see below), separated by commas.

#### logging_rft property

SHALL cause the maximum file size written to be the given value. If a file grows past this size, SHALL be closed, renamed and a new file opened.

#### logging_path property

SHALL override the TANGO_LOG_PATH environment variable and SHALL only apply to classes derived from the DServer class. The value SHALL specify where log files are written for Logging Target "file" (with no filename).

### 12.5.2 Logging Targets

TLS SHALL allow Logging Targets, that is, destinations, for messages. A Device Server SHALL emit messages to all targets (comma-separated) to its logging_target Property.

#### console

output SHALL be delivered to stdout

#### file

output SHALL be written to a standard location in a directory specified by TANGO_LOG_PATH if nonblank, otherwise in a directory created under /tmp/tango-*.

#### file::filename

output SHALL be written to the filename given. File format SHALL be log4j compatible.

#### device::devicename

output SHALL be sent to the Tango Device given by the device name given. The The Device running as devicename SHALL implement the Log Consumer interface (see below).

### 12.5.3 Logging Levels

TLS SHALL support Logging Levels. The Logging Level is meant to act as a filter for the amount of inform. These are the logging levels, with typical meanings for when they are to be used.

- OFF: Nothing is logged
- FATAL: A fatal error occurred. The process is about to abort
- ERROR: An (unrecoverable) error occurred but the process is still alive
- WARN: An error occurred but could be recovered locally
- INFO: Provides information on important actions performed

- DEBUG: Generates detailed information describing the internal behavior of a device

Levels SHALL be ordered the following way:`DEBUG < INFO < WARN < ERROR < FATAL < OFF`For a given device, a level SHALL be said to be enabled if it is greater or equal to the logging level assigned to this device. In other words, any logging request which level is lower than the device's logging level is ignored.

The all the Device's Logging Targets SHALL share the same device logging level.

### 12.5.4 C++ API

TLS SHALL provide the following C++ macros for generating messages.

There SHALL be two forms, printf-like and stream-like.

printf-like: `LOG_DEBUG(("Msg#%d - Hello world", i++));`

stream-like: `DEBUG_STREAM << "Msg#" << i++ << "- Hello world" << endl;`

These two logging requests are equivalent. Note the double parenthesis in the printf version.

The examples just given are for the DEBUG log level. There SHALL BE equivalent macros for the other log levels, except OFF.

TLS SHALL provide a means to log in the name of a device. This SHALL be provided when the Device must is a member of a class inheriting from the `Tango::LogAdapter` class.

### 12.5.5 Python API

TLS SHALL provide the folliwng Python forms of writing log messages:

basic logging: `self.debug_stream("read_voltage(%s, %d)", self.host, self.port)`

logging with print: `print >>self.debug_info, "read voltage attribute"`

logging with decorators: `@PyTango.DebugIt()`

The examples just given are for the DEBUG log level. There SHALL BE equivalent macros for the other log levels, except OFF.

### 12.5.6 Java API

TLS SHALL provide configuration to allow Java to log messages in the following manner:`get_logger().debug(Initializing device + get_name());`The example just given is for the DEBUG log level. There SHALL BE equivalent macros for the other log levels, except OFF.

### 12.5.7 Log Consumer

When a Device Target is specified as "device", this SHALL designate a Device Server which is implementing the Log Consumer interface. To be a Log Consumer, a Tango Device SHALL implement just one Tango Command:`void log (Tango::DevVarStringArray details).`

The array of strings composing the argument SHALL be interpreted with the following meanings:

- details[0] : the timestamp in millisecond since epoch (01.01.1970)
- details[1] : the log level
- details[2] : the log source (i.e. device name)

- details[3] : the log message
- details[4] : the log NDC (contextual info) - Not used but reserved
- details[5] : the thread identifier (i.e. the thread from which the log request comes from)

### 12.5.8 GUI for viewing logs

TLS SHALL include a GUI application called LogViewer or displaying logs pushed by the devices to the Tango logging system. LogViewer SHALL have the ability to fetch logs from different Tango devices and filter them by several properties. The LogViewer SHALL have the following abilities:

- show logging messages from multiple devices simultaneously
- filter logging messages by Logging Level
- be able to read log files from disk

LogViewer SHALL have two modes: static and dynamic.

- static: LogViewer SHALL be started with the name of the Log Consumer Device name and it SHALL receive all messages from Devices who have this name as their target.
- dynamic: LogViewer SHALL let the user use the GUI to specify which devices LogViewer will monitor.

## 12.6 References

The TANGO Control System: Reference Section A.1.5: The device logging https://tango-controls.readthedocs.io/en/latest/development/advanced/reference.html#the-device-logging

The TANGO Control System: Reference Section 8.3 The Tango Logging Service https://tango-controls.readthedocs.io/en/latest/development/device-api/device-server-writing.html#the-tango-logging-service

A.8 Tango log consumer https://tango-controls.readthedocs.io/en/latest/development/advanced/reference.html#tango-log-consumer

LogViewer Extension https://tango-controls.readthedocs.io/en/latest/tools-and-extensions/built-in/logviewer/logviewer.html

CHAPTER

THIRTEEN

# 15/THE DYNAMIC ATTRIBUTE AND COMMAND

## 13.1 Preamble



The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 13.2 Goals

This specification describes the Tango kernel dynamic Attribute and Command API.

## 13.3 Use Cases

The primary use case is to allow Tango Device Server programmers to create Attributes and/or Commands in runtime.

Reasons for that:

- create Attributes/Commands based on underlying library;
- create generic Tango device classes which interfaces are entirely/partly unknown at the moment of creation.

Consider a Tango device class that represents a serie of detectors. Each detector differs a bit in terms of supported features, while the most part of the code could be the same for each detector. These small differentiations may be implemented as dynamic Attributes or Commands.

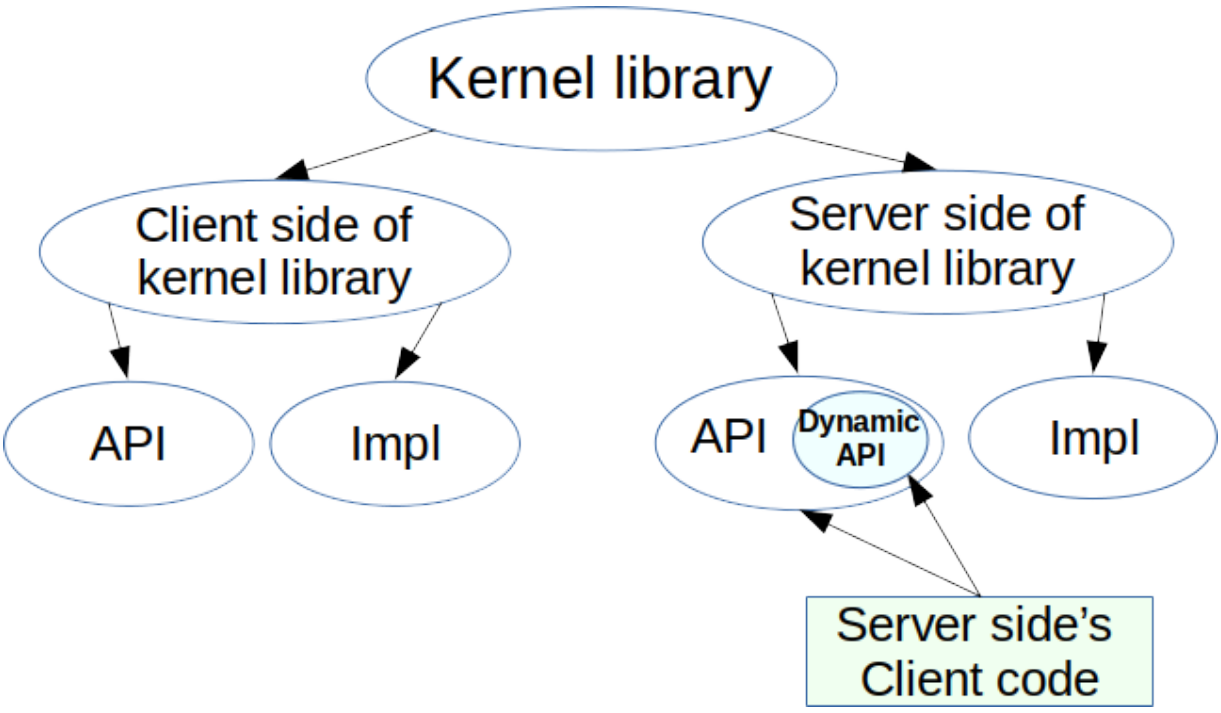
Consider a Tango Device Class that represents a residual gaz analyzer. The device will provide measurement results for many different masses. The results can be presented as a Tango spectrum attribute showing the measurements for all the masses but it can be also very interesting to create an attribute for each different mass, then the user could monitor this specific mass or define some alarm thresholds or configure archiving events for this specific mass. Dynamic attributes may be used to create easily these numerous attributes whose behaviour is almost identical.

Consider a Tango Device my/simu/device simulating the interface of another Tango Device my/original/device, for tests purposes. my/simu/device could query the interface of my/original/device at runtime and create dynamically the same attributes and commands as my/original/device. The simulated device could then be used to test high level client applications which would then talk to the simulated device instead of the original device.

## 13.4 Definitions

Tango kernel library - library that provides Tango client and server APIs

Tango server client - client code to Tango kernel library's server API




## 13.5 API that allows to create Attributes and Commands in runtime

Tango kernel library MUST provide API to server client to add new Attributes, Commands.

This API MAY accept configuration for each Attribute or/and Command.

Accepted configuration MUST be the same as described in RFC-3 (The command model) and RFC-4 (The attribute model)

As the result of the call, API creates corresponding Attribute/Command fully compliant to RFC-4/RFC-3.

Implementation of the API MAY send an Event (INTERFACE_CHANGE, see RFC-12) in response to a new Attribute/Command creation/removal.

> In Tango V10 All Commands and Attributes SHOULD be defined as dynamic i.e. use this API. High level API MAY provide sense of static entities.