TANGO Box V9

Virtual Machine User Manual

Date: October 2015

Version: 1.0

Table of Contents

Table of Contents

References

[1] Virtual Box (https://www.virtualbox.org/wiki/Downloads

[2] TANGO Box V9.0 http://sourceforge.net/projects/tango-cs/files/)

[3] PyTANGO (http://www.esrf.eu/computing/cs/tango/tango_doc/kernel_doc/pytango/latest/index.html)

[4] iTango Console (http://www.esrf.eu/computing/cs/tango/tango_doc/kernel_doc/pytango/latest/itango.html)

Introduction

Control systems for large scientific and industrial facilities are highly complex systems. They control hundreds or thousands of devices of many different types, through a variety of interfaces. Good monitoring and control is mission critical to these facilities.

The TANGO control system is a free, open source control system that is being collaboratively developed between the ALBA, ANKA, DESY, Elettra, ELI-BEAMS, ELI-ALPS, ESRF, FRM II, INAF, MAX-IV, Solaris and Soleil Institutes, with the specific aim of managing this complexity in a contemporary and powerful fashion.

Tango is object-oriented and distributed. This makes it both flexible and developer friendly.

The purpose of the Tango Box Virtual Machine is to give you a fast, out-of-the-box experience of a working Tango system. There is a limited set of shortcuts to essential Tango tools on the virtual machine desktop. Together with the introductory movie video and this user manual, they allow you to experience the power and elegance of the Tango system first hand. After this "guided tour" of the Tango system, Tango Box is an excellent tool to make further explorations on your own, to use it for demonstration purposes, to make studies, proof-of-concepts and the like. This way, out of this little box, another great, sophisticated control system for the real world gets maybe born!

Overview of Tango (Box)

TANGO consists of a number of main parts: one is a software bus, or TANGO kernel. It is a system that lets *TANGO clients* monitor and control equipment through *TANGO device servers*. This communication between clients and servers can be both synchronous, asynchronous or event driven.

Another part is the central services. A **configuration database** contains all configuration data (runtime attribute and property values that differ between instances of the same device type/class, do not belong in code, and need to be managed in central repositories for these large systems to be maintainable). An **archiving service** allows historic values of the attributes to be stored and retrieved. A **snapshot tool** allows a system state to be saved to persistent storage and to restore that state to the machine. An **alarm system** signals parameter values approaching or reaching values outside normal operating ranges and potentially damaging to the machine. Finally a diagnostic **logging** system logs diagnostic messages for later analysis.

A third major part is a set of tools that allow administration of the TANGO system. It is through these tools that we will experience the running TANGO system on the Tango Box VM. **Jive** is TANGO's database browser, providing hierarchic access to TANGO's Servers, Devices and Classes. **Astor** is the *Tango Manager* for controlling, starting and stopping TANGO device servers. **AtkMoni** is a tool for charting and monitoring device variables (attributes). A complete list is in Table 1: Summary of key TANGO Box tools.

A set of tools worth mentioning in its own right is **Sardana**. Sardana is a *standard generic user environment* built on top of TANGO, specifically for realizing control applications that require sequencing of fast and slow controls, e.g. beamlines and their experiments. It is exposed through the **Spock** and **iTango** shortcuts on the Tango Box desktop.

Setting up your System

Installing Virtual Box

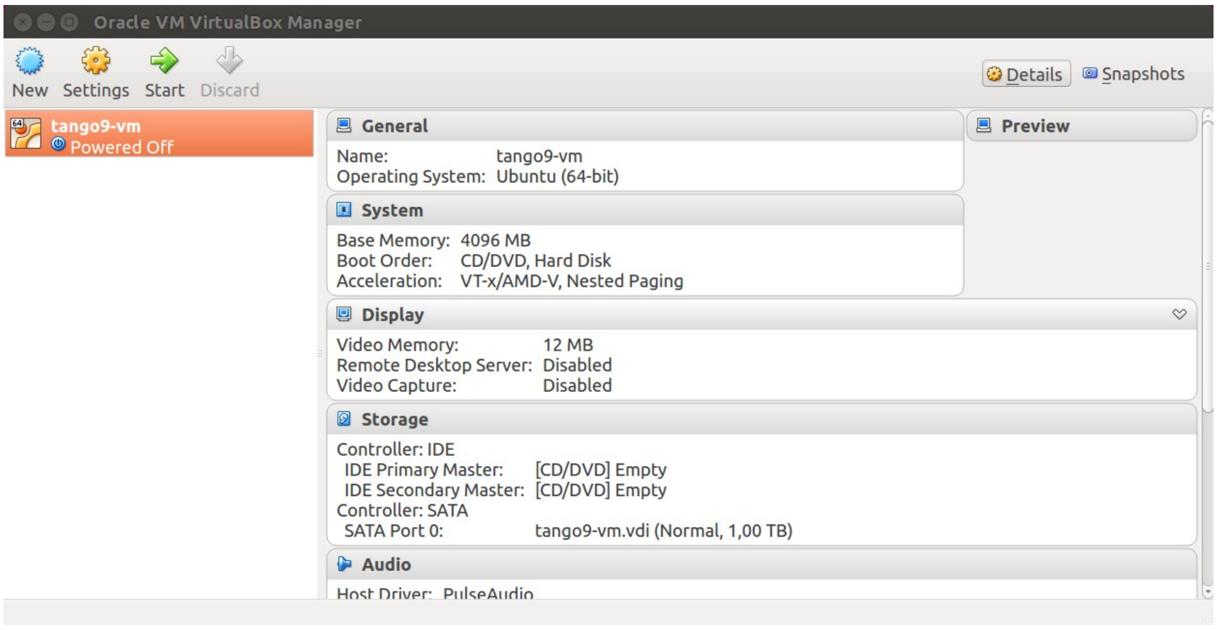
TANGO Box is a virtual machine that is based on the Ubuntu operating system and the easiest way to run TANGO Box is to use *Virtual Box* which allows you to run images of different operating systems on your existing system. Download *Virtual Box* (https://www.virtualbox.org/wiki/Downloads) and install according to the instructions provided.

Download TANGO Box

The TANGO Box image which is run in Virtual Box can be downloaded from the TANGO website TANGO Box V9.0 http://sourceforge.net/projects/tango-cs/files/).

Note: The extraction from the archive can take several minutes.

Running TANGO Box



tango9-vm
Powered Off

General
Name: tango9-vm
Operating System: Ubuntu (64-bit)

System
Base Memory: 4096 MB
Boot Order: CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging

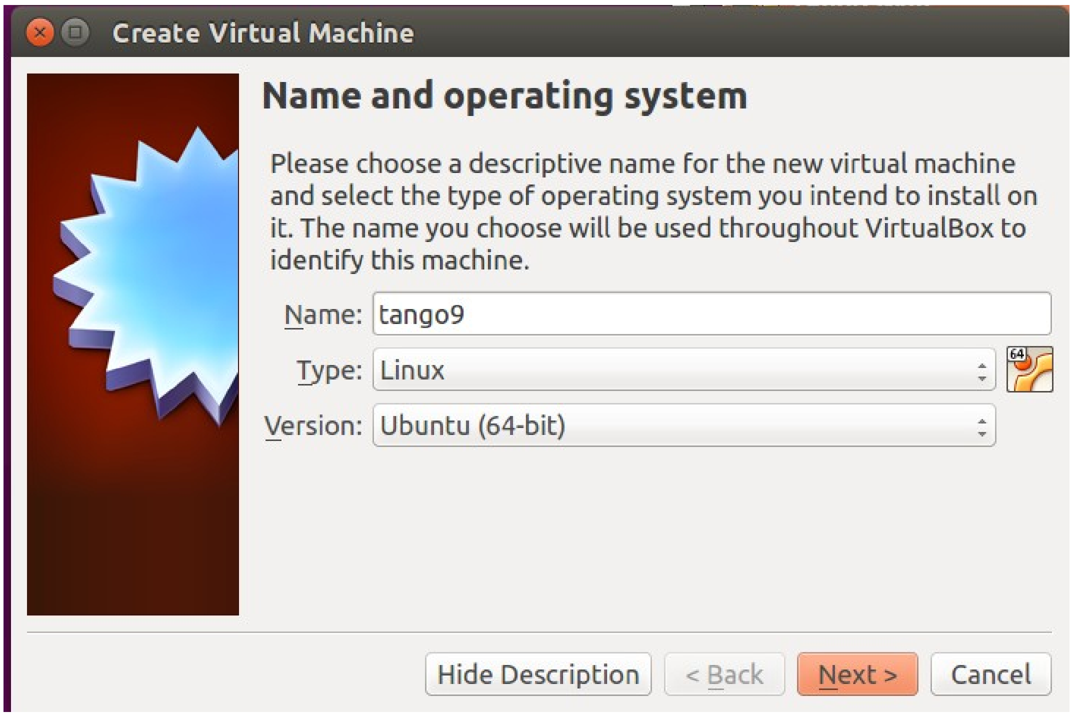
Display
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage
Controller: IDE
IDE Primary Master: [CD/DVD] Empty
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: tango9-vm.vdi (Normal, 1,00 TB)

Audio
Host Driver: PulseAudio

Figure 1: Virtual Box opening screen

1. Launch Virtual Box → New Virtual Machine.
2. Choose Linux, Ubuntu 64 bit and name e.g. tango9 :



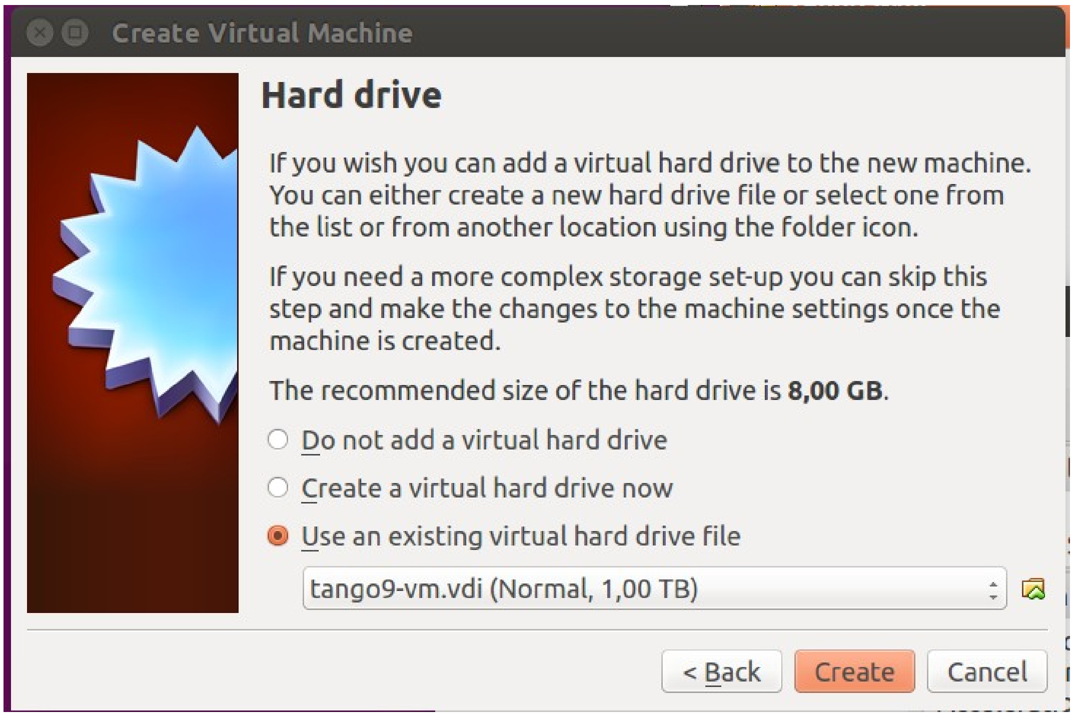
Name and operating system

Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: tango9
Type: Linux
Version: Ubuntu (64-bit)

3. Select settings according to your machine – recommended memory setting of 4 MB

4. Select the disk image you download and unzipped – tango9-vm.vdi :



Create Virtual Machine

Hard drive

If you wish you can add a virtual hard drive to the new machine. You can either create a new hard drive file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

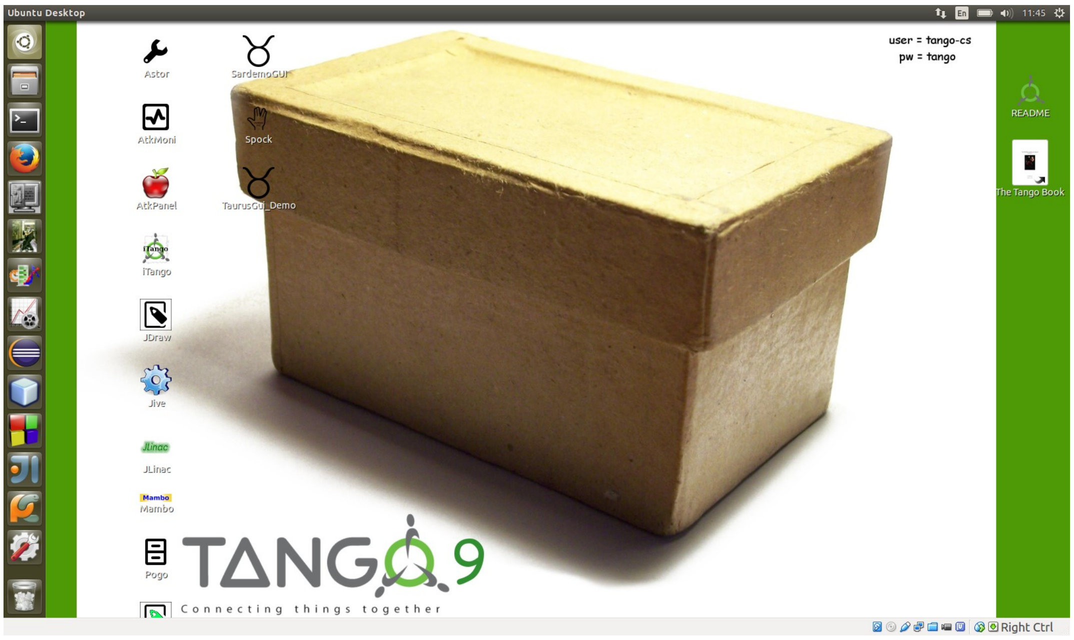
The recommended size of the hard drive is **8,00 GB**.

- Do not add a virtual hard drive
- Create a virtual hard drive now
- [✓ Use an existing virtual hard drive file]

tango9-vm.vdi (Normal, 1,00 TB)

< Back | Create | Cancel

5. Start the TANGO9 Virtual Machine and discover the wonderful world of TANGO9 :



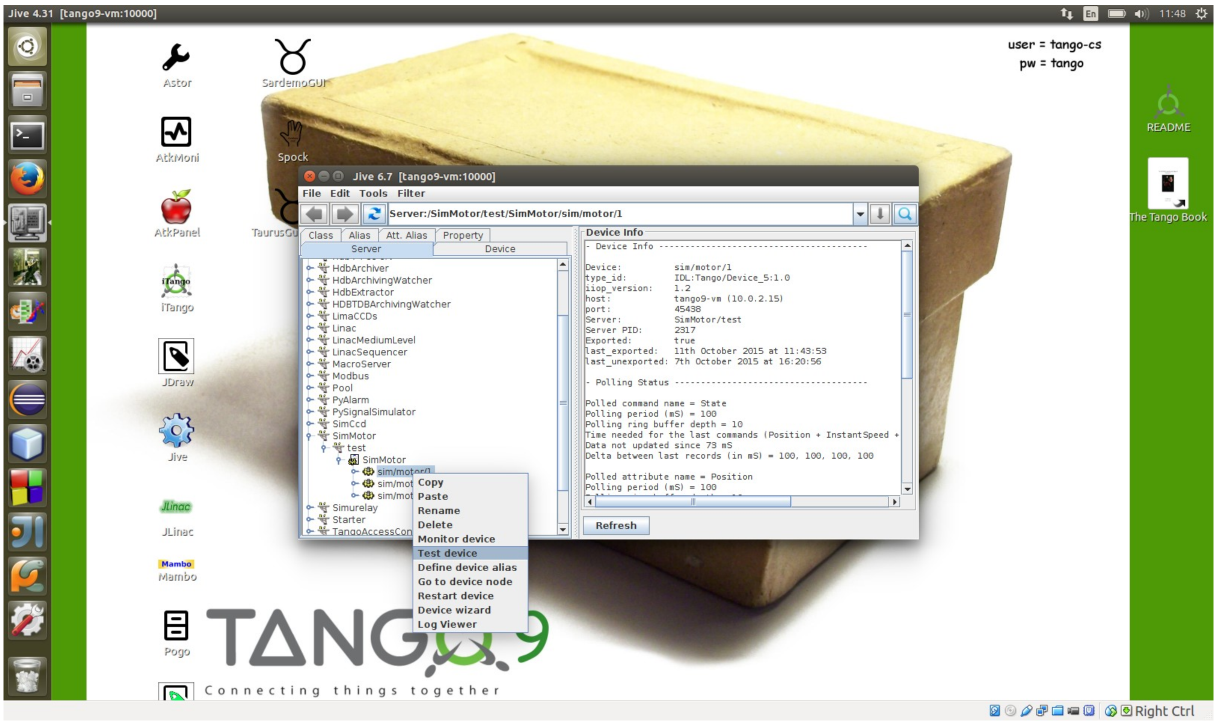
Whetting your Appetite

A Quick Out-of-the Box Experience

Experiencing a TANGO System running a Motor Simulator

Using Jive

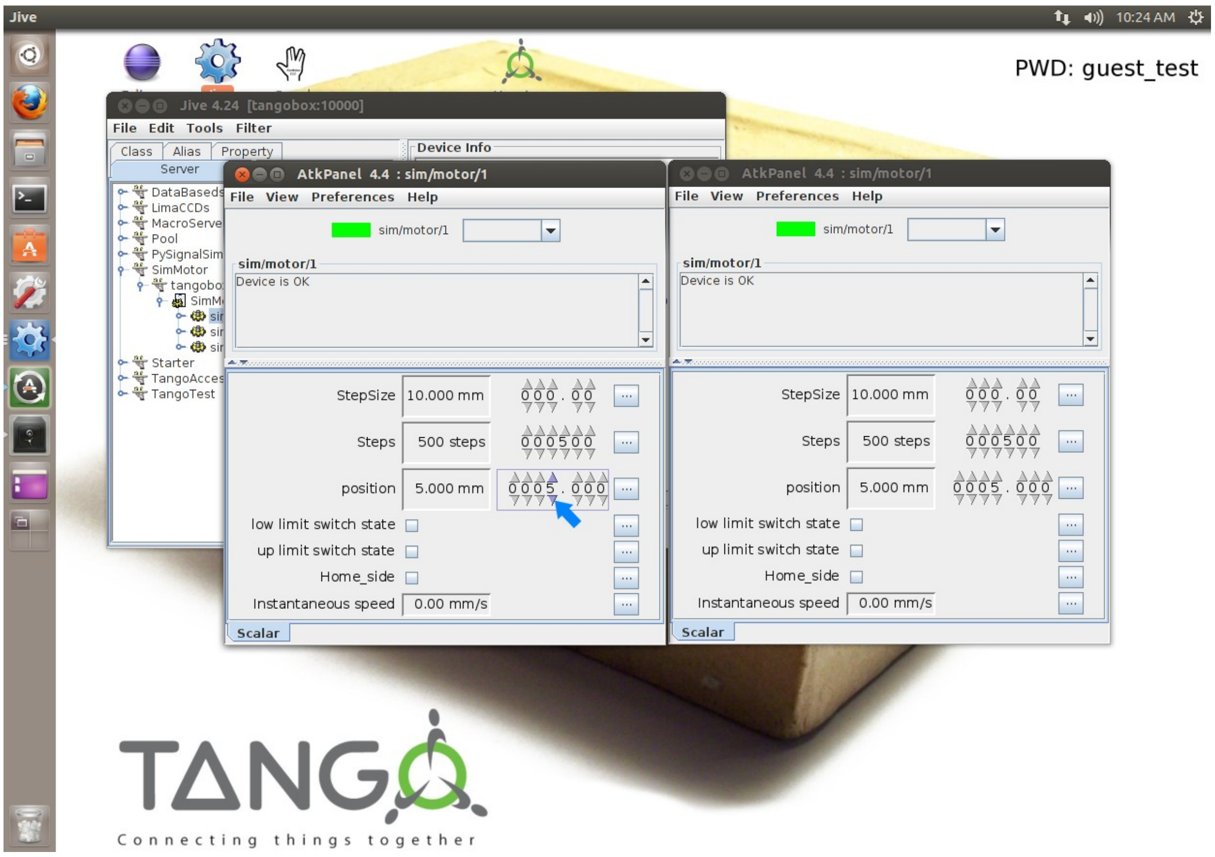
1. Launch *Jive* from the Desktop.



```
- Device Info -----------------------------------------

Device:          sim/motor/1
type_id:         IDL:Tango/Device_5:1.0
iiop_version:    1.2
host:            tango9-vm (10.0.2.15)
port:            45438
Server:          SimMotor/test
Server PID:      2317
Exported:        true
last_exported:   11th October 2015 at 11:43:53
last_unexported: 7th October 2015 at 16:20:56

- Polling Status -------------------------------------

Polled command name = State
Polling period (mS) = 100
Polling ring buffer depth = 10
Time needed for the last commands (Position + InstantSpeed +
Data not updated since 73 mS
Delta between last records (in mS) = 100, 100, 100, 100

Polled attribute name = Position
Polling period (mS) = 100
```

Figure 3: Jive screen showing hierarchy of device server (SimMotor), device server instance (tangobox), device class (SimMotor) and device class instance (sim/motor/1)

2. Click on SimMotor → TANGObox.
3. Right click on sim/motor/1 and select Monitor Device.
4. Repeat above to open a second "Monitor Device" window for sim/motor/1.
5. This opens the auto-generated device client.



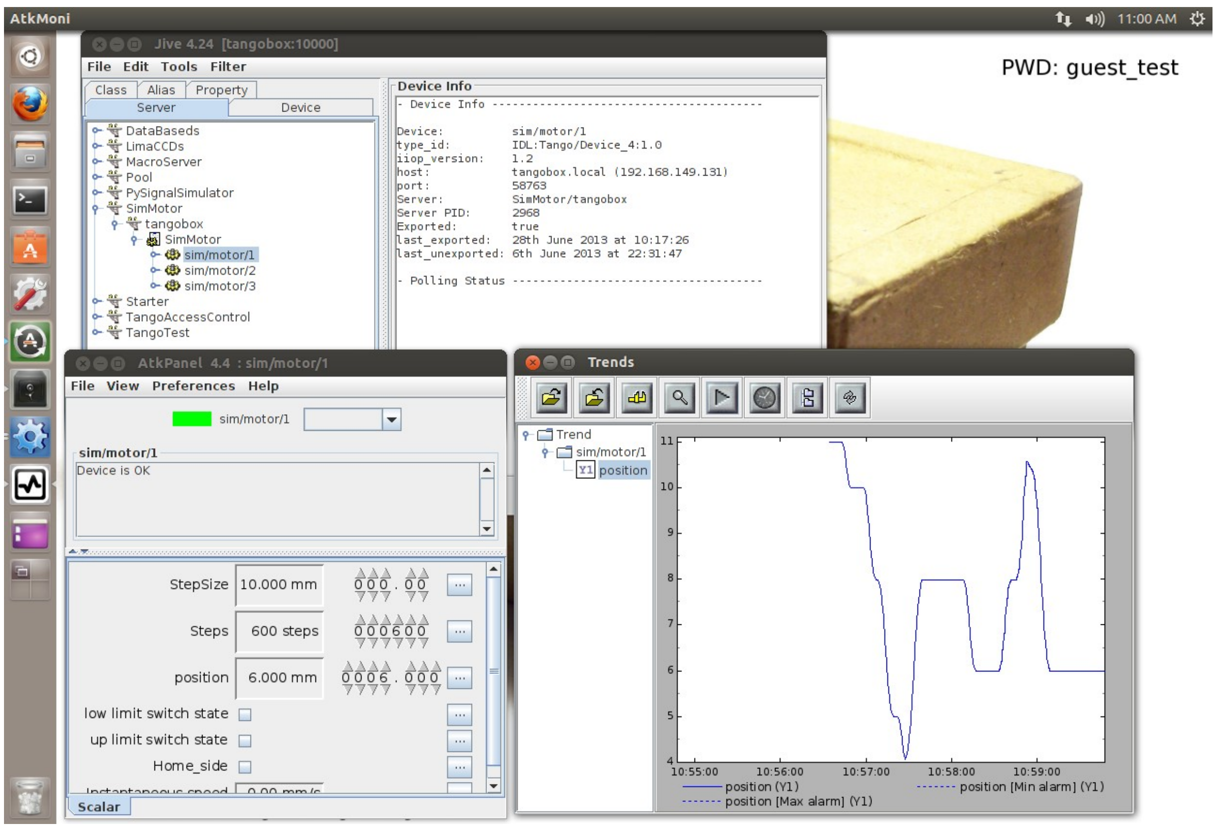
sim/motor/1

Device is OK

StepSize	10.000 mm	000.00
Steps	500 steps	000500
position	5.000 mm	0005.000
low limit switch state	☐	
up limit switch state	☐	
Home_side	☐	
Instantaneous speed	0.00 mm/s	

Figure 4: AtkPanel showing attributes for sim/motor/1. The position of the motor can be changed by using the controls indicated by the blue arrow.

6. Change the motor position using the controls indicated by the blue arrow.



```
- Device Info ----------------------------------------

Device:          sim/motor/1
type_id:         IDL:Tango/Device_4:1.0
iiop_version:    1.2
host:            tangobox.local (192.168.149.131)
port:            58763
Server:          SimMotor/tangobox
Server PID:      2968
Exported:        true
last_exported:   28th June 2013 at 10:17:26
last_unexported: 6th June 2013 at 22:31:47

- Polling Status ------------------------------------
```

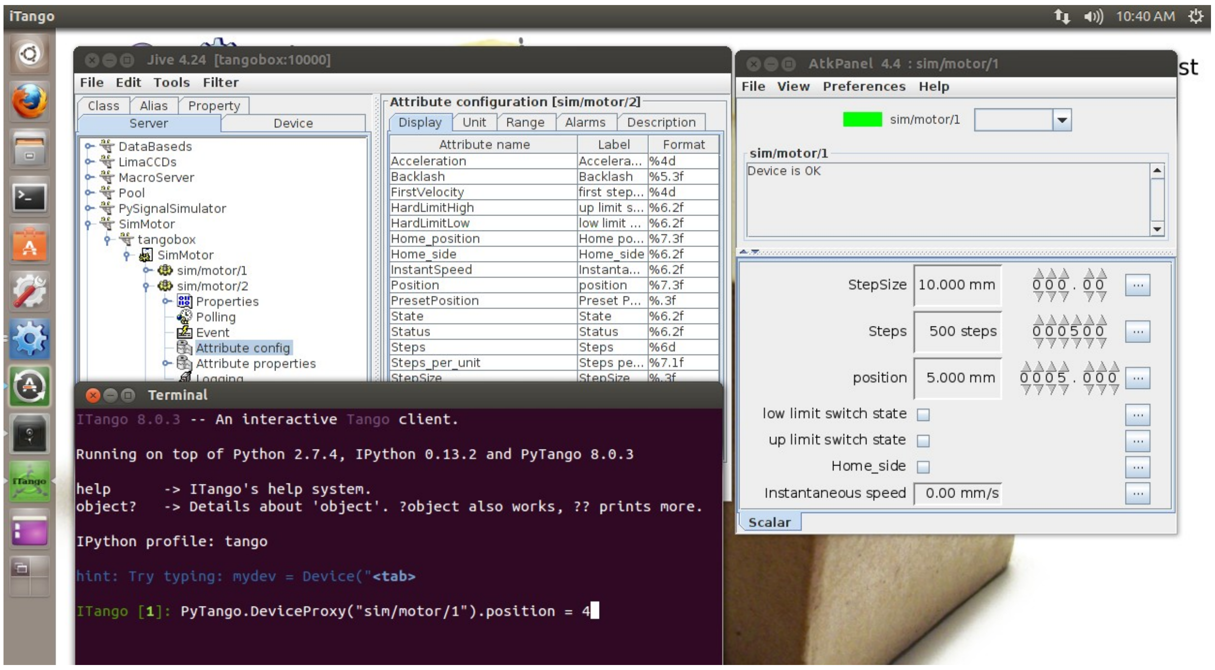
sim/motor/1
Device is OK

StepSize	10.000 mm
Steps	600 steps
position	6.000 mm

Figure 5: AtkMoni (Trends) showing the trend over time of the Temperature attribute for sim/motor/1.

7. You can view how the motor position is changing by launching *AtkMoni*. Click on Add New Attribute (toolbar button) and select in the tree: sim → motor → 1 → position. Then right-click on the position attribute and choose "Add to Y1". Then click on Start Monitoring (toolbar button). Now, if you change the position in Jive, you can see the position change in *AtkMoni*.

Using iTango



Attribute name	Label	Format
Acceleration	Accelera...	%4d
Backlash	Backlash	%5.3f
FirstVelocity	first step...	%4d
HardLimitHigh	up limit s...	%6.2f
HardLimitLow	low limit ...	%6.2f
Home_position	Home po...	%7.3f
Home_side	Home_side	%6.2f
InstantSpeed	Instanta...	%6.2f
Position	position	%7.3f
PresetPosition	Preset P...	%.3f
State	State	%6.2f
Status	Status	%6.2f
Steps	Steps	%6d
Steps_per_unit	Steps pe...	%7.1f
StepSize	StepSize	%.3f

sim/motor/1
Device is OK

StepSize	10.000 mm	000.00
Steps	500 steps	000500
position	5.000 mm	0005.000
low limit switch state	☐	
up limit switch state	☐	
Home_side	☐	
Instantaneous speed	0.00 mm/s	

```
ITango 8.0.3 -- An interactive Tango client.

Running on top of Python 2.7.4, IPython 0.13.2 and PyTango 8.0.3

help      -> ITango's help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

IPython profile: tango

hint: Try typing: mydev = Device("<tab>

ITango [1]: PyTango.DeviceProxy("sim/motor/1").position = 4
```

Figure 6: iTango terminal

1. Launch *iTango*
2. Enter (copy) the command into the iTango console.

```
PyTango.DeviceProxy("sim/motor/1").position=4
```

Motor can be seen to be moving in the ATKPanel.

Tip

When typing, try pressing <tab>. Since *iTango* has autocomplete embedded you get a list of possible completions. Example:

`PyTango.Release.<tab>`

Gets a list of all members of PyTango.Release class. Equally:

`PyTango.DeviceProxy("sim<press tab key>`

Gives a list of available runtime options
sim/motor/1   sim/motor2   sim/motor3

Note

We put the command in line of Python code to save you a copy-paste.
The following works equally well of course:

```
motor1 = PyTango.DeviceProxy("sim/motor/1")
motor1.position = 4
```

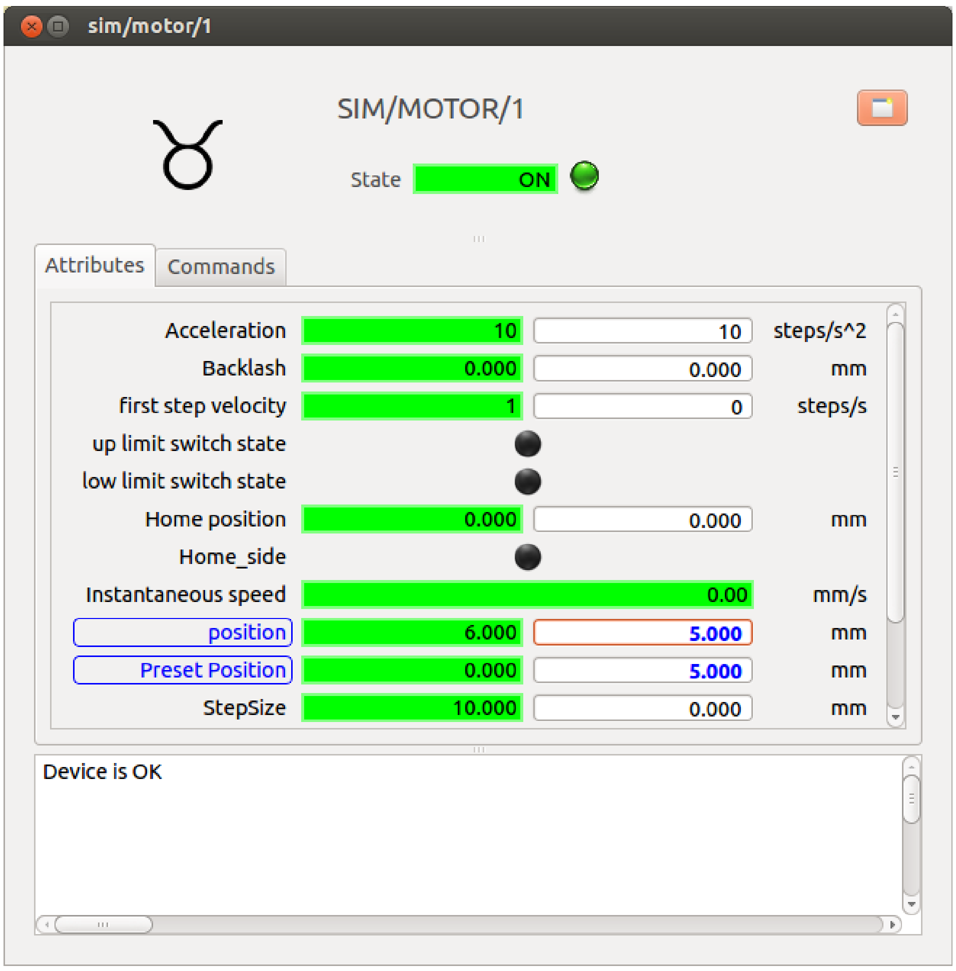
Using Taurus

As we will see further, Taurus is a library to create GUI's for the TANGO system in Python. It comes with a collection of demos. We will use one of them as a starting point for this demo.

In a new terminal window type:

```
taurusform  sim/motor/1
```

This launches the form below, which provides again access to the various motor attributes. We leave it to the reader to explore the GUI, change values and observe the synchronization via the TANGO device server of these two TANGO device clients: the ATK-based monitor panel and the Taurus based form.



State ON

Acceleration	10	10	steps/s^2
Backlash	0.000	0.000	mm
first step velocity	1	0	steps/s
up limit switch state	●		
low limit switch state	●		
Home position	0.000	0.000	mm
Home_side	●		
Instantaneous speed	0.00		mm/s
position	6.000	5.000	mm
Preset Position	0.000	5.000	mm
StepSize	10.000	0.000	mm

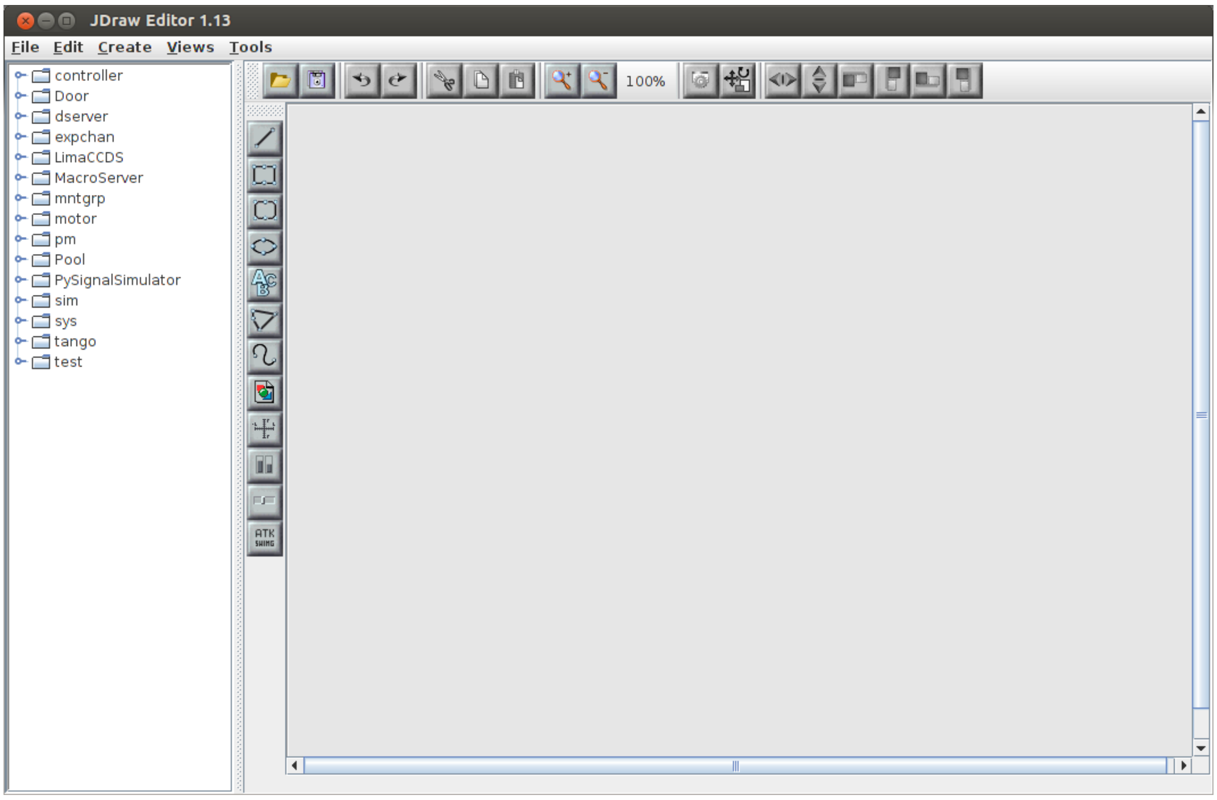
Device is OK

Figure 7: Taurus Form for sim/motor/1

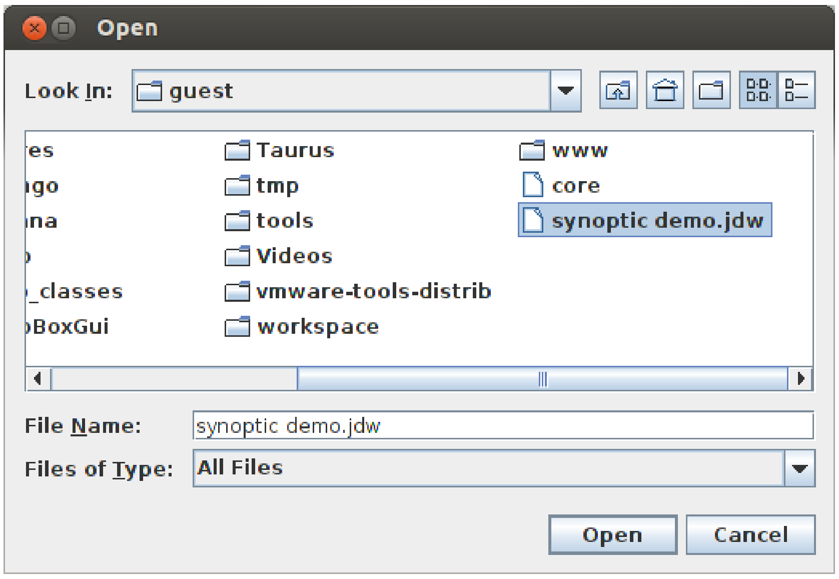
Using JDraw

JDraw is the TANGO synoptic editor. You can use to construct (draw) synoptic screens, i.e. GUI's with graphical representation of the physical layout of the device to be controlled by TANGO.

1. Launch *JDraw*



2. Open the file synoptic demo.jdw.



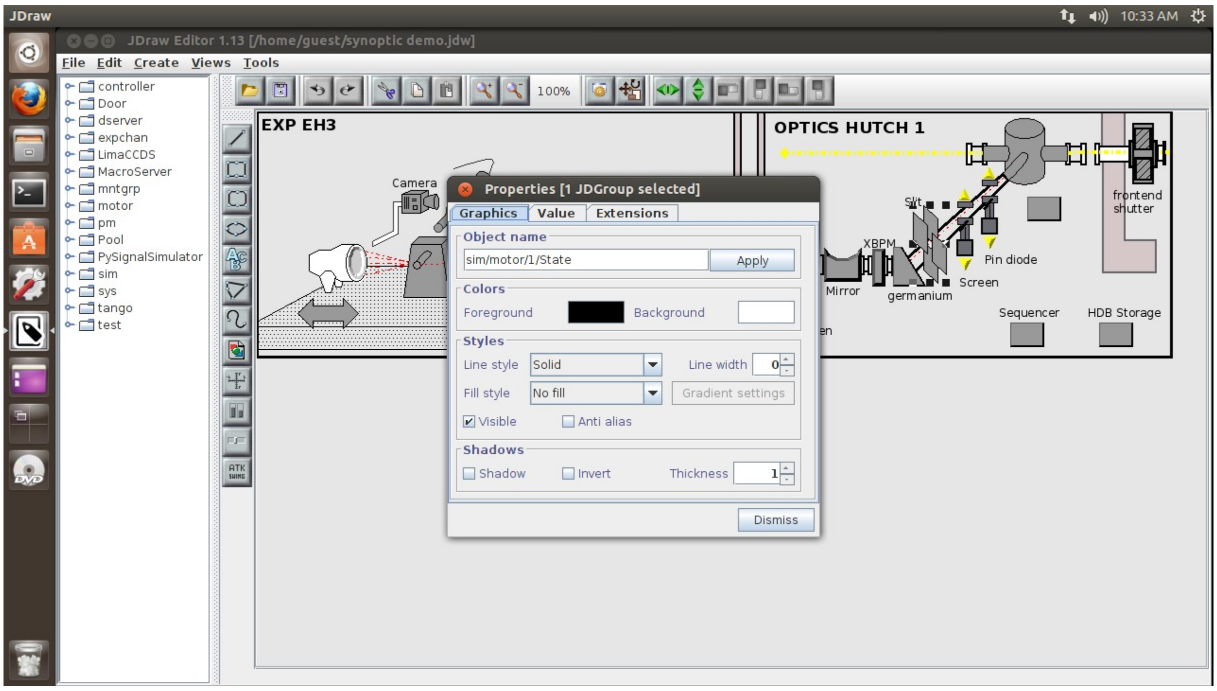
Look In: guest

File Name: synoptic demo.jdw

Files of Type: All Files

You see a graphical representation of a beam line with its optics elements. To provide a live demo of TANGO synoptic screens, the right-most slit is already associated with the State of our sim/motor/1 from the previous examples

3. You can see the associated object by double-clicking on the slit. The Object name field in the dialog that opens reads: sim/motor/1/State.



Object name: sim/motor/1/State

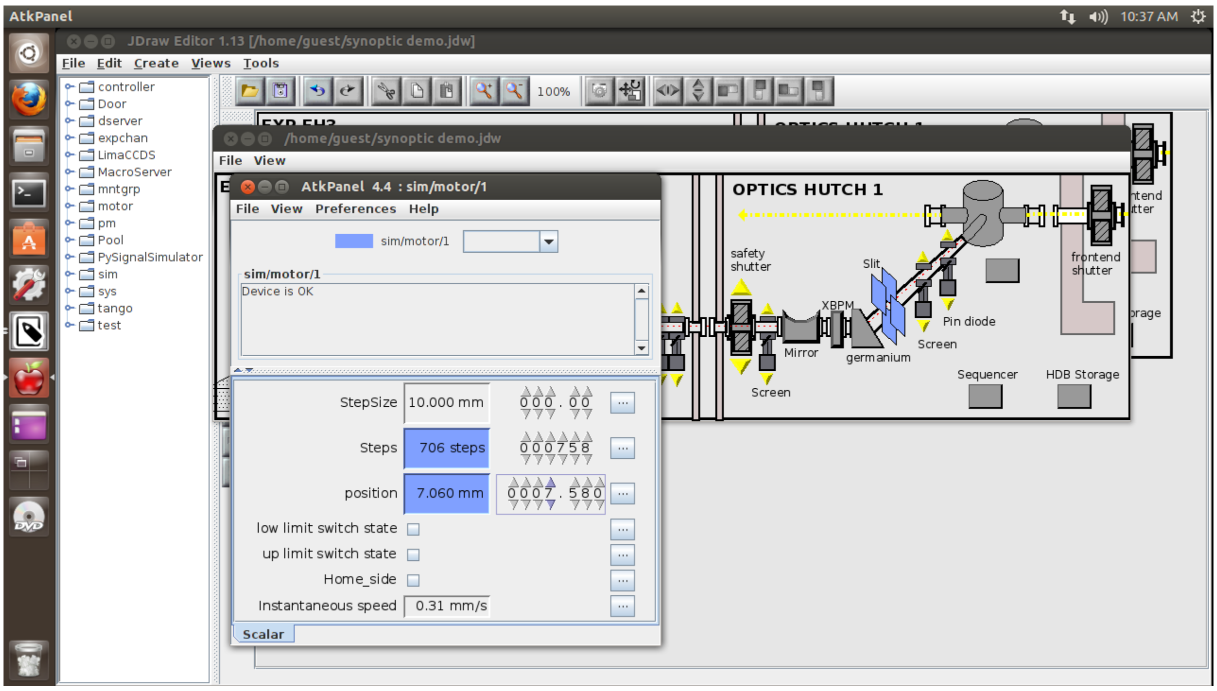
Colors — Foreground: ■ Background: □

Styles — Line style: Solid; Line width: 0; Fill style: No fill; [✓] Visible; [ ] Anti alias

Shadows — [ ] Shadow; [ ] Invert; Thickness: 1

4. Before we go to live view let us associate the leftmost slit (slit2) as well with a motor: sim/moter/2. Double-click it and type in Object name: sim/motor/2/State and click apply

5. Go to Views -> Tango Synoptic view. Now if we change the position of any of both motors in ATKPanel, we see that the slit in the synoptic view indicates that a movement (blue color) is occurring.



sim/motor/1

sim/motor/1
Device is OK

```
StepSize                 10.000 mm    000.00
Steps                    706 steps    000758
position                 7.060 mm     0007.580
low limit switch state   ☐
up limit switch state    ☐
Home_side                ☐
Instantaneous speed      0.31 mm/s
```

TANGO Box Tools Overview

Now that you have had a taste for running TANGO, we can examine some of the tools that are packaged with TANGO. These tools have shortcuts on the desktop and are summarised in Table 1.

Table 1: Summary of key TANGO Box tools

Tool	Purpose
Astor/Starter	The Astor/Starter pair of applications will help you to administer your control system (Starting/Stopping device server, checking them, etc.). Astor is a graphical application and Starter is the name of the device server used by Astor.
ATK	TANGO Application ToolKit (ATK) is a java graphical layer for building GUIs for TANGO.
ATKPanel	ATKPanel is a generic application which displays panels allowing you to execute any device commands or to read/write any device attributes
ATKMoni	AtkMoni is the tool to view how your data is changing.
Eclipse	Widely used development environment
iTango	iTango is a PyTANGO CLI based on IPython. It is designed to be used as an IPython profile.
JDraw	Synoptic editor to draw your synoptic(s) to control TANGO devices (To be used with ATK)
Jive	The TANGO database browser and device testing tool
Netbeans	Alternate development environment
Pogo	Allows you to create/update TANGO device classes in C++
Spock	Spock is an iTango based CLI (command line interface) for Sardana. Spock has been extended in Sardana to provide a customized interface for executing macros and automatic access to Sardana elements.
TaurusDesigner	Powerful TANGO client GUI designer. It is a Qt designer application customized for taurus

The Elegant TANGO Architecture: A Quick Overview

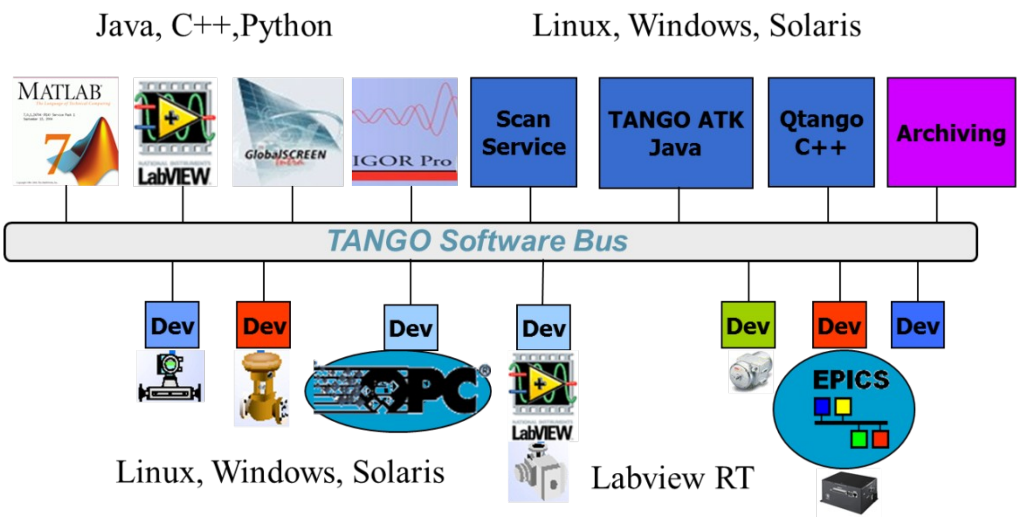


Figure 8: Overview of the TANGO Architecture

The following provides a short description of the main architectural elements of the TANGO system to provide the system context. In the next chapter we look into more detail into those elements that are most relevant to the programmer/user of the TANGO system: the OO aspects.

TANGO is based on the concepts **of object oriented** and **service oriented** approaches to software architecture. The object model in TANGO supports methods, attributes and properties. In TANGO all objects are representations of devices.

TANGO is primarily used to provide **network access to hardware**. Hardware access is programmed in a process called a Device Server. The device server implements device classes which implement the hardware access. At runtime the device server creates **devices** which **represent logical instances** of **hardware**. Clients "import" the devices and send requests to the devices using the TANGO protocol.

The object model in TANGO supports methods, attributes and properties. In TANGO all objects are representations of devices. The devices can be on the same computer or distributed over a number of computers interconnected by a network. The network communication is done using **CORBA**. Communication can be **synchronous**, **asynchronous** or **event driven**. **Configuration data** is stored in a **database**. Programming support is provided for C**++, Java and Python**. Clients can be written in all three languages. Servers can also be written in C++, Java or Python. TANGO provides a **kernel API** which **hides** all the **details** of **network access** and provides **object browsing**, **discovery** and **security features**.

Some ready to use **graphical applications** (DeviceTree, ATKPanel, …) allow you to graphically display data coming from your device(s). **Graphical layers** above the kernel API have been developed to reduce specific graphical client software development time. One exists for Java SWING (ATK), another and another one for Python PyQt (Taurus).

TANGO uses **CORBA** (synchronous and asynchronous communication) and **zeromq** (event based communication) and is a distributed control system that can run on one or many machines.

TANGO uses the **omniorb** implementation of CORBA as its network protocol. The client-server model is the basic communication model. Communication between clients and servers can be synchronous, asynchronous or event driven.

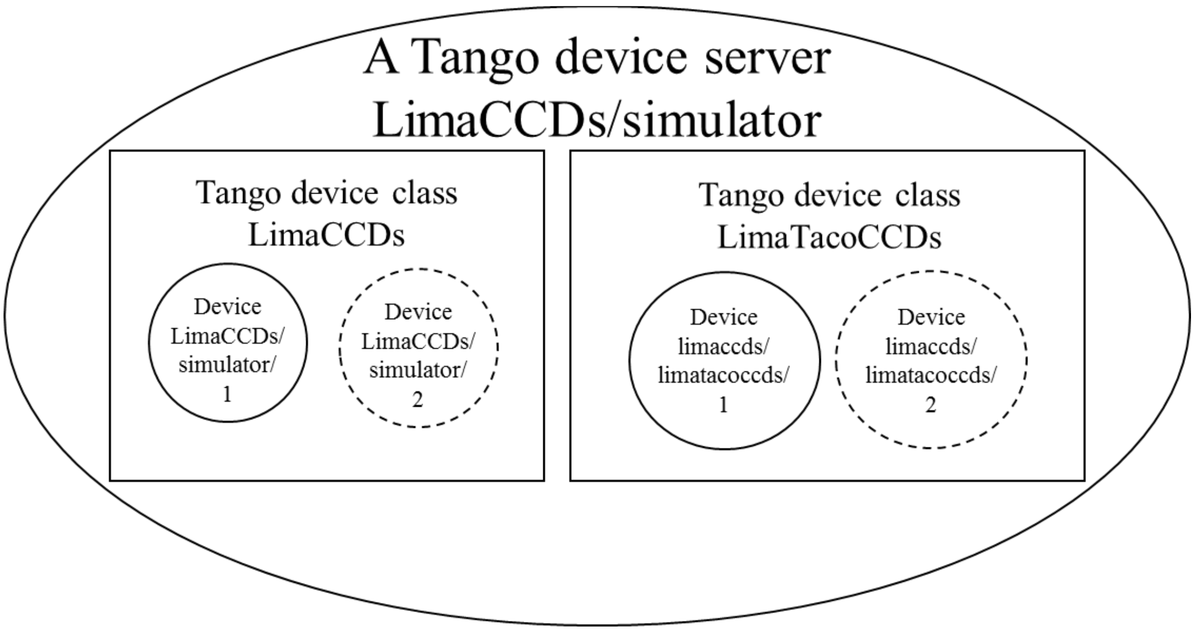
TANGO is Programmer Friendly!

It is Object Oriented

TANGO uses concepts (& constructs) from object oriented programming languages to help manage the inherent complexity of large control systems.

For example there will be many instances of the same piece of hardware equipment in the entire system, for example magnet power supplies. Where at runtime the currents they supply will be different, they obviously share many properties that are the same (e.g. you interface to them in the same way). In OO this static commonality is captured in a *Class* and the differences in run-time *objects* or *instantiations* of that class. That is exactly what TANGO provides for us. **TANGO Device Classes** and **TANGO Device Instances**. The SimMotor that we saw in the Wetting Your Appetite chapter is an example of a Device Class, "sim/motor/1", the name of a Device Instance.

Being able to build hierarchies of things (objects) of specific types is another powerful concept of the OO world. A TANGO system has such a hierarchy and it supports its distributed nature. Here we introduce the concept of a **TANGO Device Server** (DS). Essentially the DS is the process in which Device Instances (of Classes) are running. The TANGO system allows setting it up in this flexible way:


A Tango device server
LimaCCDs/simulator
Tango device class
LimaCCDs
Device LimaCCDs/ simulator/ 1
Device LimaCCDs/ simulator/ 2
Tango device class
LimaTacoCCDs
Device limaccds/ limatacoccds/ 1
Device limaccds/ limatacoccds/ 2

So, one Device Server (process) can run multiple instances of multiple device classes!

(Note: the dotted line around the second instance is because in your Tango Box, there is only one instance at start-up, but there could be more)

If we apply this to our example of the multiple power supplies, we see TANGO offers us many ways to set up this system, depending on the physical layout and other parameters of the machine. If two power supplies are close to each other, we can control them from one PC (crate), with one device server process, or with two device servers, on the same host machine. If they are far from each other, we could control them with two device servers on two separate host machines. TANGO is not putting any restrictions on our system configuration here. And for the TANGO clients, this is all transparent. An important consequence is also we can redesign the physical layout of the system, without a big impact on the software clients.

Device Attributes, Properties, Commands and State

Encapsulating data and behaviour into single entities is another important OO concept; it is an improvement over how you can model, represent the real-world in your code, compared to older programming techniques.

TANGO uses these concepts to represent the physical equipment the system is controlling. But as we will see it goes beyond having the data in data members and the behaviour invoked through methods: A Tango device class has additional provisions that are essential in the domain of control systems.

First there is the distinction between attributes and properties. In general OO terminology they are used interchangeably. In TANGO, the attribute is used to represent process variable, physical values like current, voltage, temperature, … they are supposed to vary continuously. Properties are much more static, they are the things you need to set up once for the software device to work, to connect properly to the physical equipment etc. Typical example is a IP address.

Attributes have a set of domain-specific meta-data. Just to name a few important ones: the physical unit (Ampere, mm, …), range values, alarm values.

Commands are the "dials and knobs" of the device. In a distributed system it is essential that you can distinguish between synchronous and asynchronous calls. TANGO supports that.

A last OO concept we mention is the "state" of the class, i.e. technically is that combined values of all data members. In control systems it is worth to be more specific, to model devices as state machines with a set of predefined states and transitions between those states. TANGO provides that: by default, TANGO classes have a state attribute for this purpose and there is a set of predefined states: ON, OFF, OPEN , CLOSE, MOVING, …

Let us conclude the brief introduction on the OO concepts used in TANGO here. The TANGO architecture relies heavily on more advanced concepts and techniques of the OO world, such as design patterns, but a discussion of this is outside of the scope of this VM User Manual. Please refer to the following sources for more information:

Abstract device pattern and Tango, A.Götz e.a.

Tango Control System Manual v 8.1: 8.1.2 The Device Pattern

Python powered tools: iTango, Sardana & Spock

Programmer friendliness is also greatly enhanced if you can interface dynamically with a system through an interpreted, high-level programming language such as Python.

Simple and powerful GUI building

Taurus Designer is a powerful TANGO client GUI designer. It is based on QtDesigner, and extended with additional Taurus widgets and functionality to serve for the purposes of TANGO development. With this tool, it is possible to assemble and configure a TANGO client GUI, without the need to modify the code itself.

In this section, we will construct such a simple client that will include most common Taurus widgets for extracting and writing data from and to a device.

1. Open Taurus Designer by typing in a terminal window:

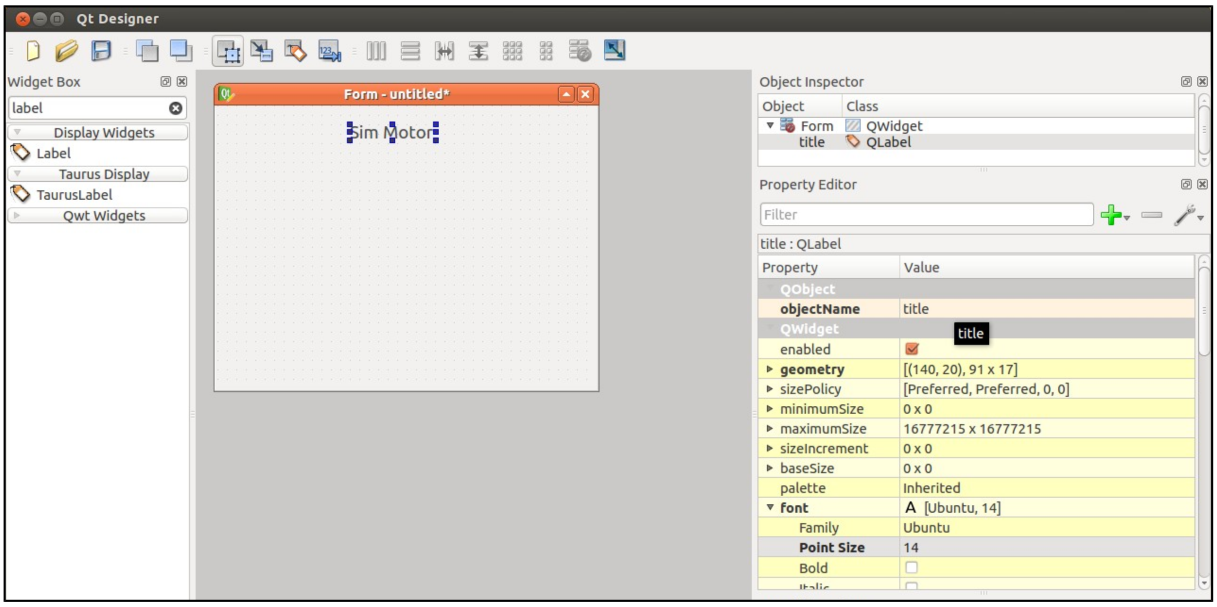
```
taurusdesigner
```

2. Create New Form

In the first window, you can select a type of the form that will be used. You can choose from preassembled templates, widgets or any custom user widgets. In the "templates/forms" section, select a "Widget" and create it.

3. Adding a new Label for Title

A standard Label is used for displaying a String. On the left, in the "Widget Box" section, under "Display Widgets", you will find a "Label" widget. Drag it onto your form. Position it, double click it and set the string to display a desired message/title. In the property editor on the right, you can modify it furthermore. Change the "objectName" to a meaningful name, e.g. "title" and modify the font, e.g. to font size 14.



Widget Box filter: label

Object	Class
Form	QWidget
title	QLabel

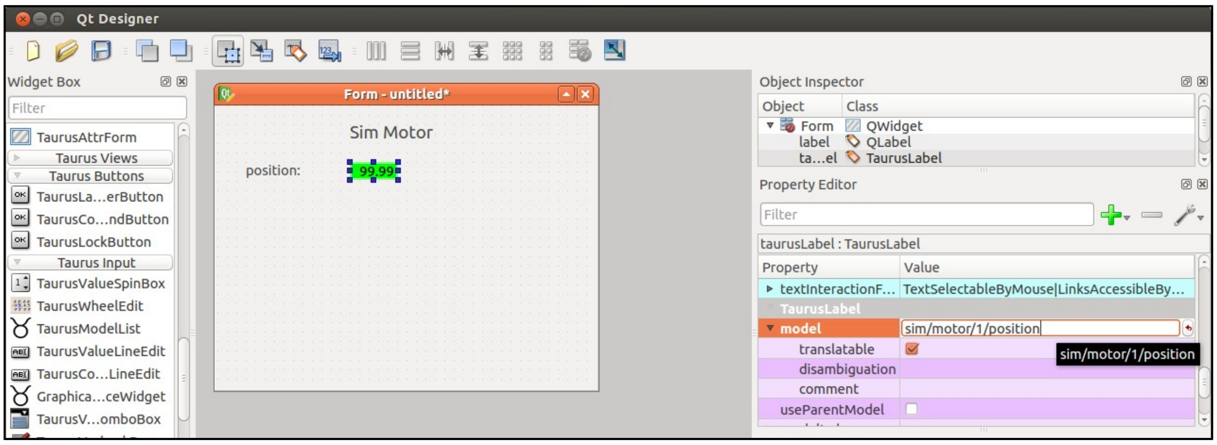
title : QLabel

Property	Value
QObject	
objectName	title
QWidget	
enabled	✓
geometry	[(140, 20), 91 x 17]
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	0 x 0
maximumSize	16777215 x 16777215
sizeIncrement	0 x 0
baseSize	0 x 0
palette	Inherited
font	A [Ubuntu, 14]
Family	Ubuntu
Point Size	14
Bold	☐

4. Attribute Readout

Create a new Label for displaying the name of the attribute.

Create a new TaurusLabel for displaying the actual value of the attribute. You will find it in the Widget Box under Taurus Display. Drag it onto the form. In order to connect this field to an actual device, all you have to do is to specify the device and the attribute. In the Property Editor, change the value of “model” to “sim/motor/1/position”, where “sim/motor/1” defines a device and “position” defines the attribute of the device.



Object	Class
Form	QWidget
label	QLabel
ta...el	TaurusLabel

taurusLabel : TaurusLabel

Property	Value
textInteractionF...	TextSelectableByMouse|LinksAccessibleBy...
TaurusLabel	
model	sim/motor/1/position
translatable	✓
disambiguation	
comment	
useParentModel	☐

5. State Indicator using TaurusLed

Create a new Label for displaying a name of the attribute (State).

Create a new TaurusLed. In the property Editor, change its value of “model” to “sim/motor/1/state”, where again “sim/motor/1” defines the device.

6. Status Readout

Repeat the “Attribute Readout” part for an attribute “status”

7. Attribute Input

Here, you will create a spin box, used for writing a value to an attribute of the device.

Create a new TaurusValueSpinBox (first of the Taurus Input widgets). In the Property Editor, change its value of “model” to “sim/motor/1/position”, where “sim/motor/1” defines the device and “position” defines the attribute.

8. Command Button

Here, you will create a button that will execute a command on the device.

Create a new TaurusCommandButton. In the property Editor, set the value of “model” to “sim/motor/1”. This defines a device. Change the value of “command” to a name of a command, for instance “StepDown”.

9. Generate Code

After saving the *.ui file in the designer, execute the command for generating the python code.

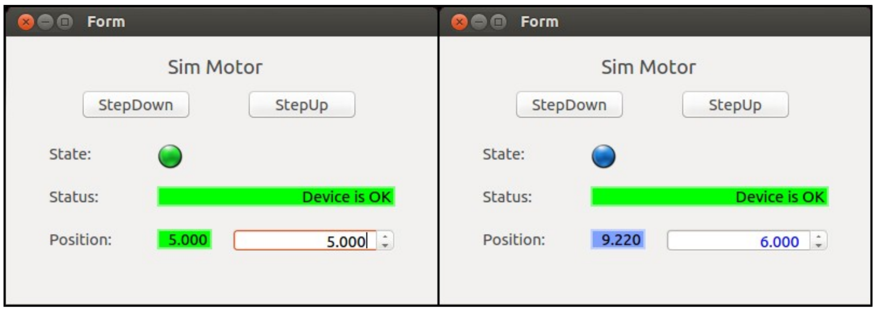
```
cd /ProjectPath
taurusuic4 -x -o ui_name.py fileName.ui
```

10. Execute the code

```
python ui_name.py
```

11. Testing the client

If everything was properly set, your GUI client should look similar to:



Try changing the position. Execute commands using buttons. Validate if the position, State and Status are properly read and displayed.

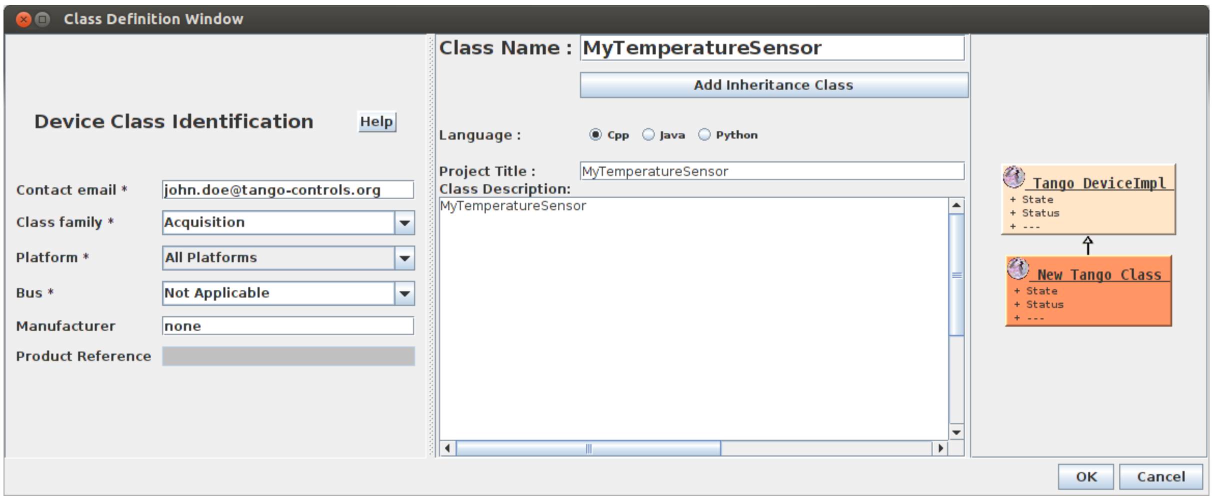
Putting it into Practice: Writing a Simple Device Server

Let us dive a bit deeper and create an actual Tango Device Class and Device Server in C++, using the Pogo code generator.

For this example we will be creating a simple temperature sensor. Start by launching *Pogo* from the Desktop.

Enter the data as in the screenshot

1. Enter Class Name (MyTemperatureSensor)
2. Project Title (idem)
3. Class Description (idem for this demo. In real world applications please describe the class behaviour here in detail)
4. Fill in Device Class Identification information (not filling them won't allow you to OK the dialog)
5. Click on Ok.



Class Name : MyTemperatureSensor
Language : [✓ Cpp] Java Python
Project Title : MyTemperatureSensor
Class Description: MyTemperatureSensor

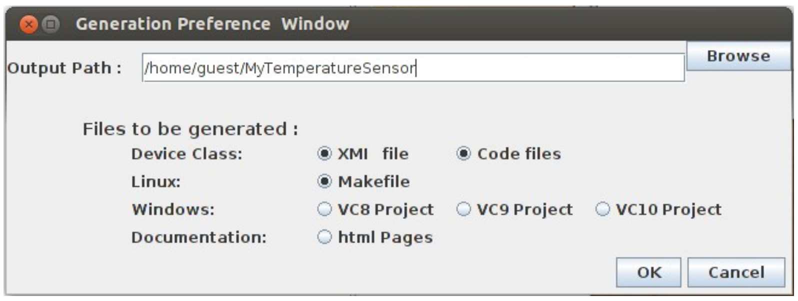
Contact email *	john.doe@tango-controls.org
Class family *	Acquisition
Platform *	All Platforms
Bus *	Not Applicable
Manufacturer	none
Product Reference	

Figure 9: Class Definition Window

Next we will add a Scalar Attribute that will represent our physical value we want to measure: Temperature. All the default values (type double, read-only, etc.) are ok for our example: click OK.

Now we can generate our C++ files. Click File→Generate or ctrl-G.

- Set Output path to /home/guest/MyTemperatureSensor
- Select Linux Makefile, so we can easily build the project


Generation Preference Window

Output Path : /home/guest/MyTemperatureSensor

Files to be generated :

Device Class: (●) XMI file (●) Code files

Linux: (●) Makefile

Windows: ( ) VC8 Project ( ) VC9 Project ( ) VC10 Project

Documentation: ( ) html Pages

Figure 10: Scalar Attribute creation and Generation Preference Window

Launch terminal

```
cd MyTemperatureSensor/

make
```

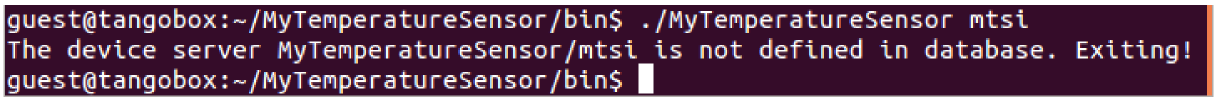
The code is compiled.

```
cd bin/
```

In the `bin` directory there is an executable `MyTemperatureSensor` executable. This is a device sever. Run the device server with an instance, `msti`, in this case:

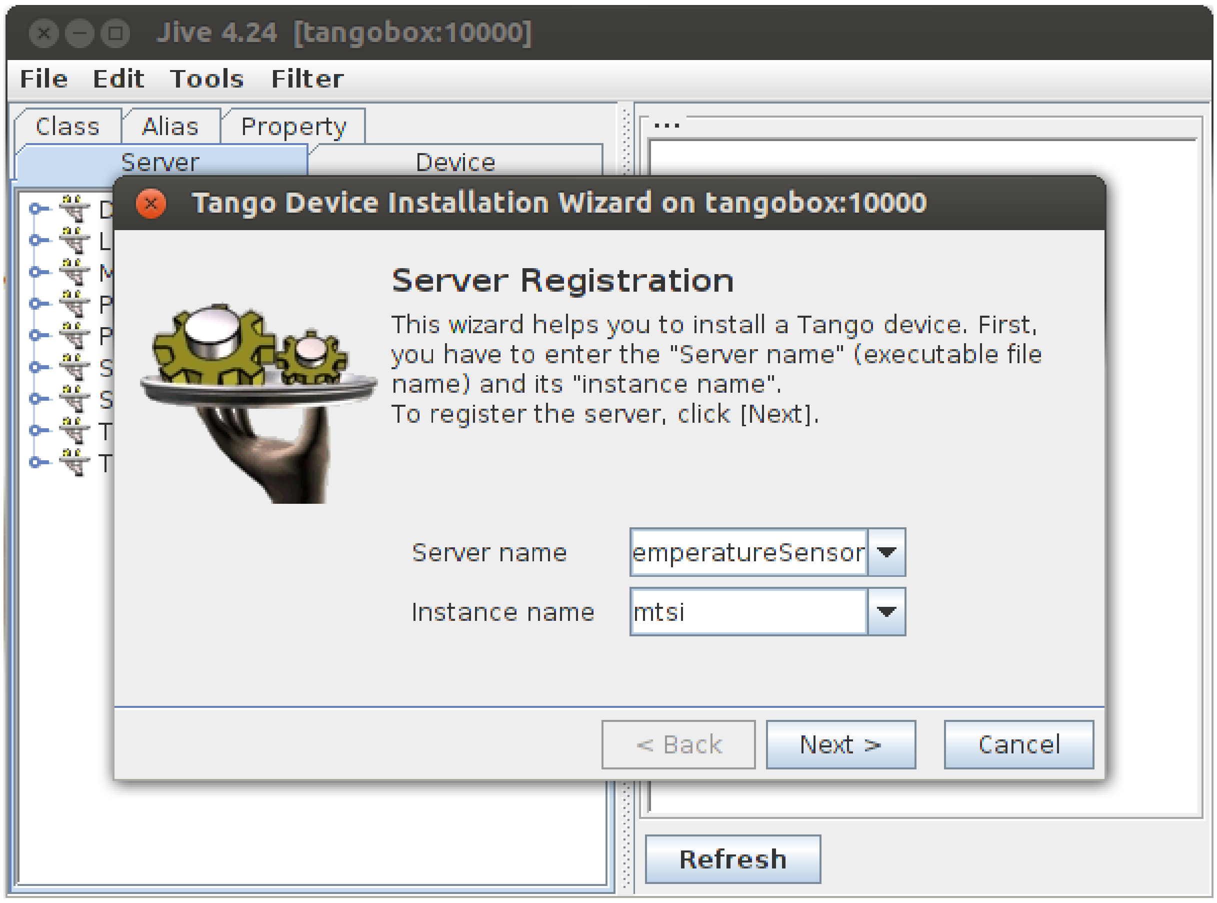
```
./MyTemperatureSensor  mtsi
```

However, the device server has not been registered so you get an error message.



```
guest@tangobox:~/MyTemperatureSensor/bin$ ./MyTemperatureSensor mtsi
The device server MyTemperatureSensor/mtsi is not defined in database. Exiting!
guest@tangobox:~/MyTemperatureSensor/bin$
```

Launch Jive and go to the Server Wizard to register the device server (MyTemperatureServer) and its instance (mtsi).



Tango Device Installation Wizard on tangobox:10000

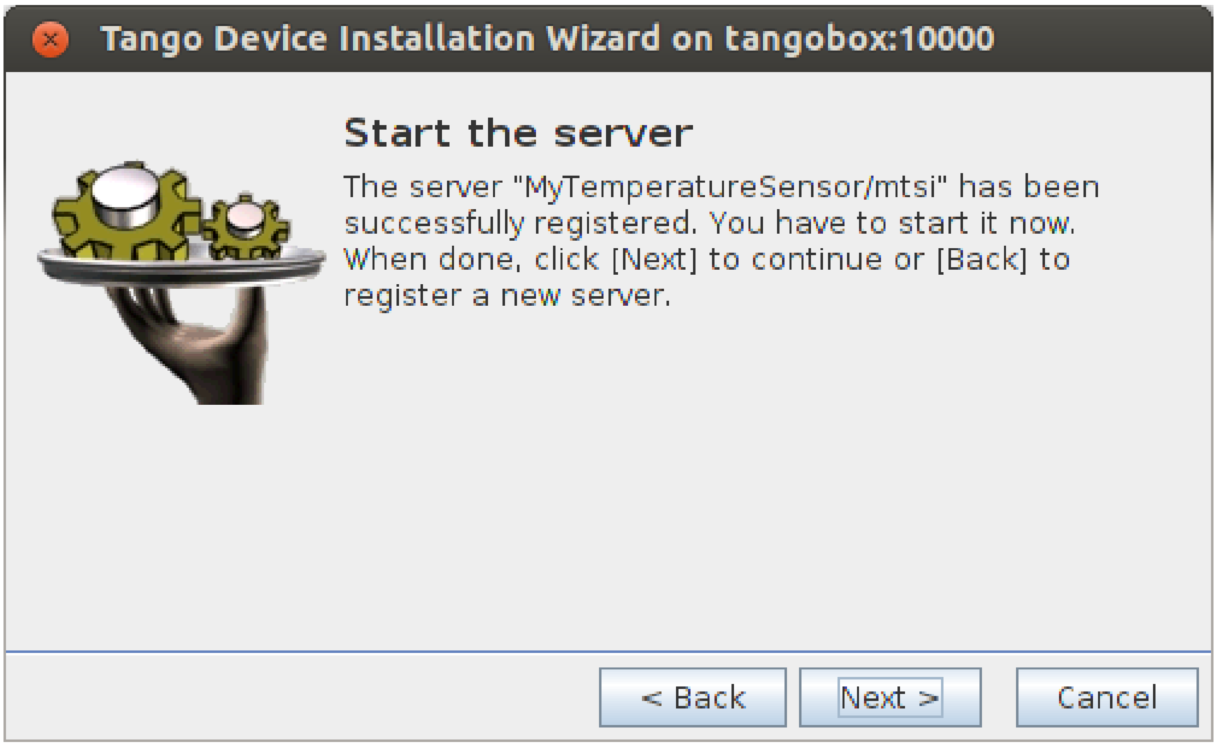
Server Registration

This wizard helps you to install a Tango device. First, you have to enter the "Server name" (executable file name) and its "instance name".
To register the server, click [Next].

Server name: emperatureSensor

Instance name: mtsi

Click Next.


Tango Device Installation Wizard on tangobox:10000

Start the server

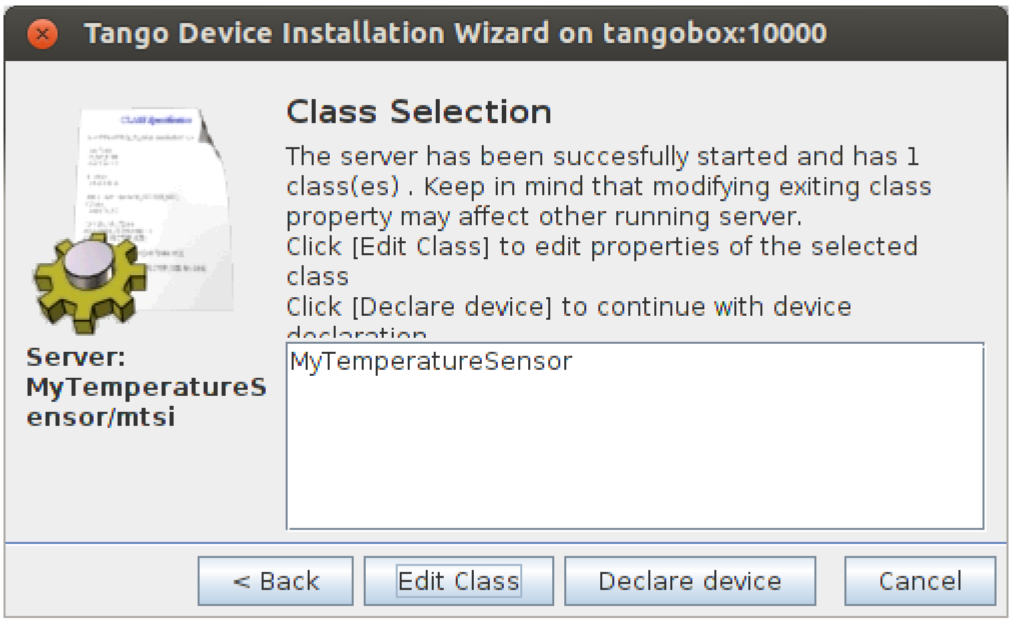
The server "MyTemperatureSensor/mtsi" has been successfully registered. You have to start it now. When done, click [Next] to continue or [Back] to register a new server.

Now is the right time to start our server:

In the terminal windows, run the device server with an instance, `msti`, in this case:

```
./MyTemperatureSensor  mtsi
```

In Jive, click next on Start the server screen



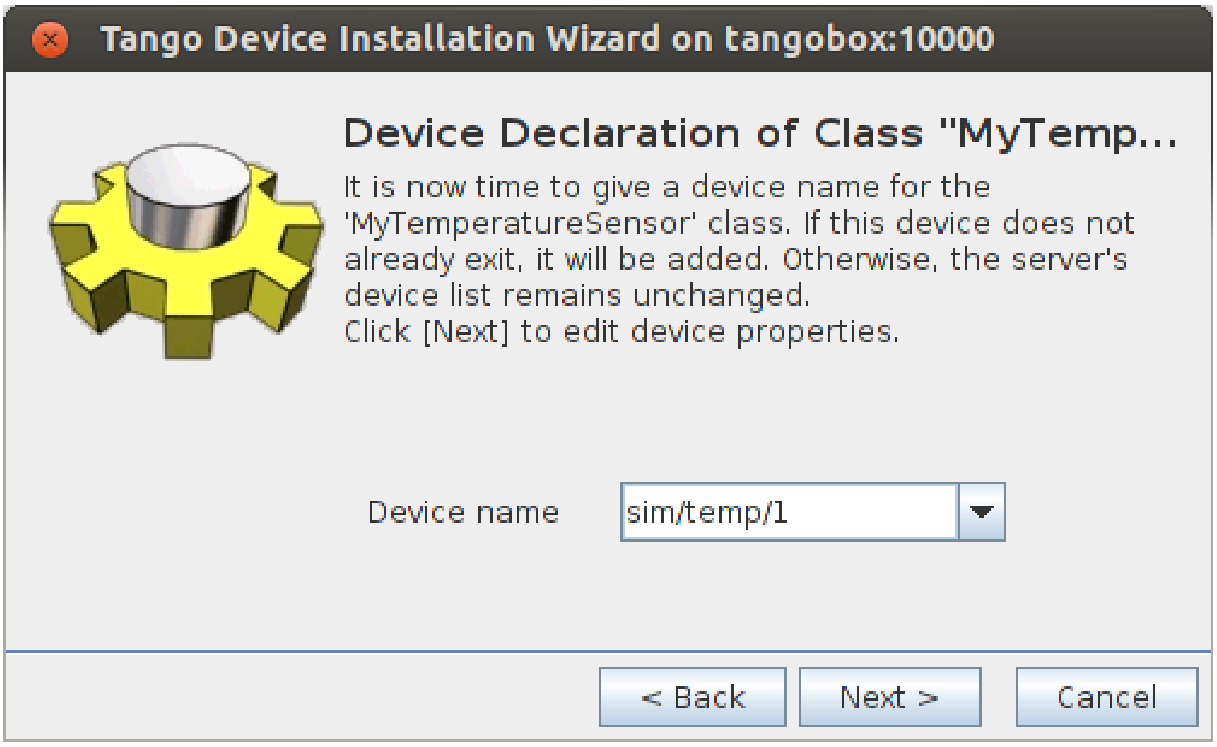
Class Selection

The server has been succesfully started and has 1 class(es) . Keep in mind that modifying exiting class property may affect other running server.
Click [Edit Class] to edit properties of the selected class
Click [Declare device] to continue with device declaration

Server: MyTemperatureSensor/mtsi

MyTemperatureSensor

To instantiate a device, we need to select its Class type. In this case there is just one Class type in our Device Server (but remember there can be more). Select MyTemperaturSensor and click Declare Device.



Device Declaration of Class "MyTemp...

It is now time to give a device name for the 'MyTemperatureSensor' class. If this device does not already exit, it will be added. Otherwise, the server's device list remains unchanged.
Click [Next] to edit device properties.

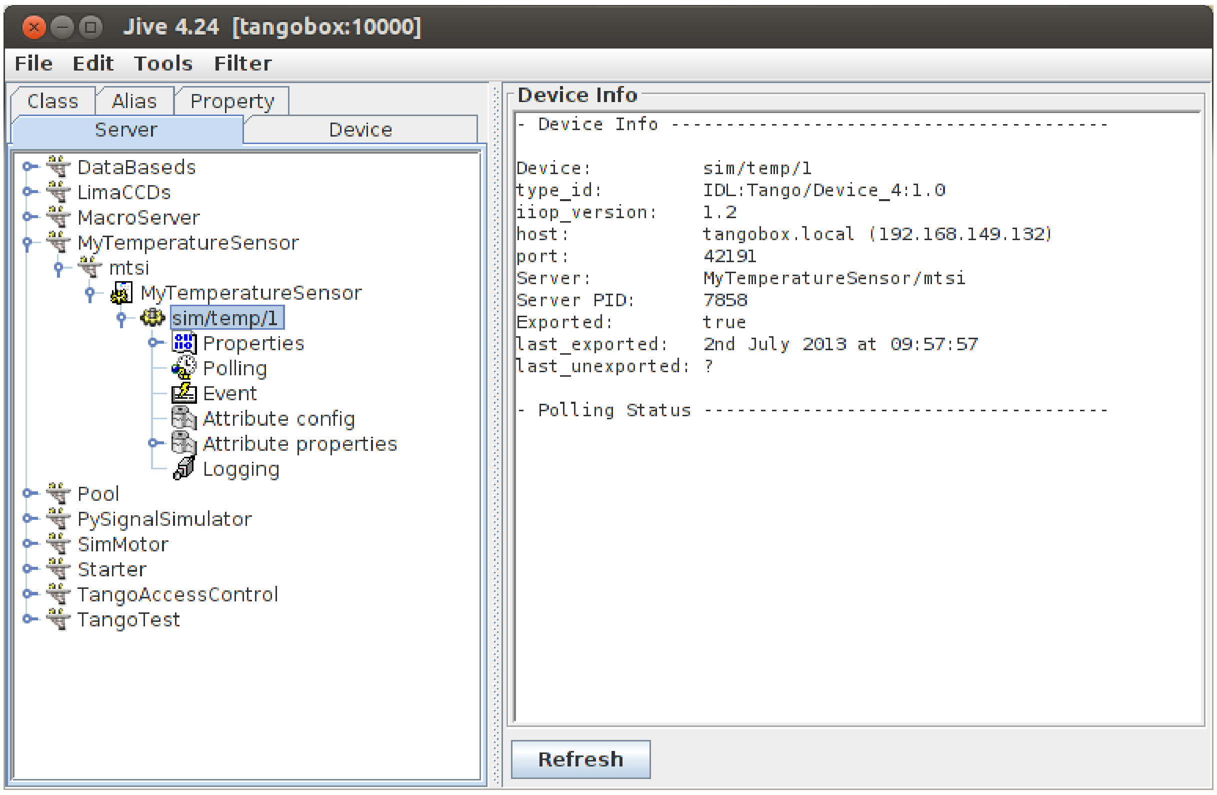
Device name sim/temp/1

1. Enter device name: sim/temp/1

Note: for the purpose of this demo you could any string consisting of 3 fields, separated by slashes. The aim of this naming system for big real-world systems is to allow you to construct a logical domain/family/member hierarchy of **Devices** that is fully independent of the physical Device Server Layout.

2. Click Next
3. Finish
4. Click Yes on "Would you like to reinitialize the server?"

`MyTemperatureSensor` now appears in list of device servers in Jive. Right click on `Temperature` and select `Monitor Device`. Temperature is 0 as expected, we declared an attribute but did not implement anything that would change its value



```
- Device Info ----------------------------------------

Device:           sim/temp/1
type_id:          IDL:Tango/Device_4:1.0
iiop_version:     1.2
host:             tangobox.local (192.168.149.132)
port:             42191
Server:           MyTemperatureSensor/mtsi
Server PID:       7858
Exported:         true
last_exported:    2nd July 2013 at 09:57:57
last_unexported: ?

- Polling Status ------------------------------------
```

Figure 11: Device server `MyTemperatureSensor` has a device class `sim/temp/1`

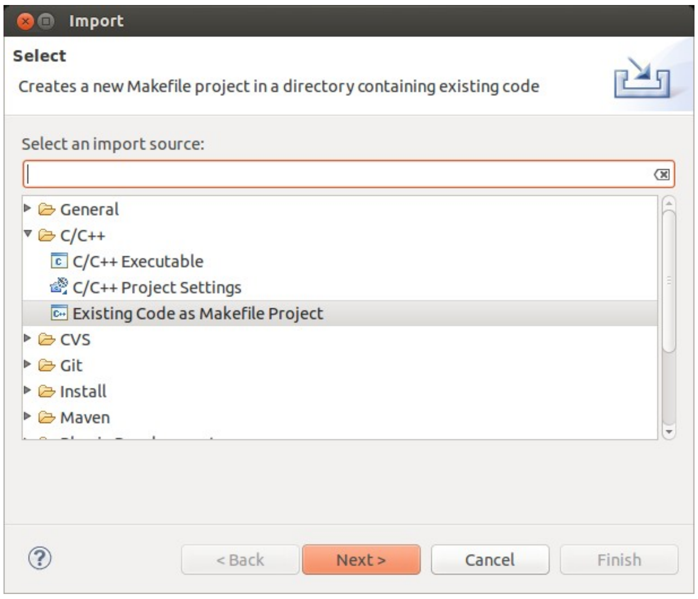
We can now modify the code to introduce some changes to the temperature and make the demo meaningful.

Before we change the code, stop the device server (Ctrl-z in the terminal where it is running).

Note	For the purpose of this demo you can keep using make and simple text editing: just navigate to the source files with *Files* and edit them with right-click→Open With→GNU Emacs. If you prefer you can skip the section on Eclipse

Launch *eclipse*.

File→Import→Existing Code as Makefile Project



Import

Select

Creates a new Makefile project in a directory containing existing code

Select an import source:

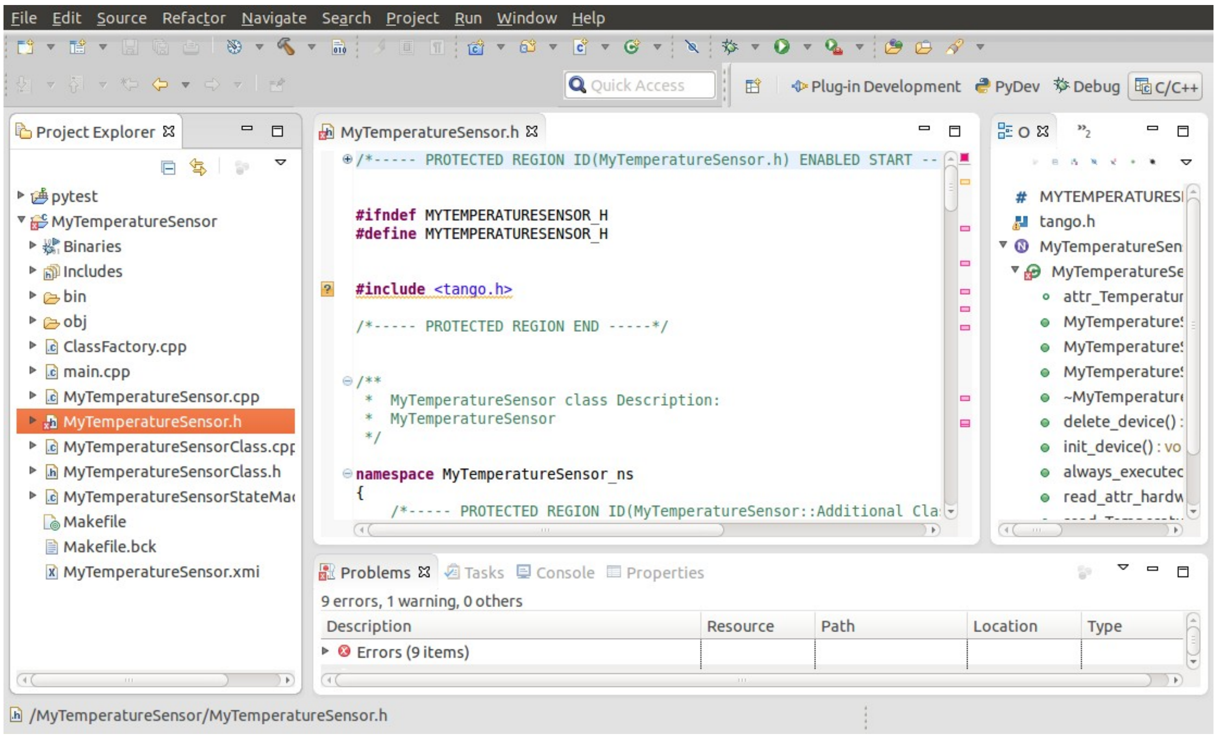
- General
- C/C++
  - C/C++ Executable
  - C/C++ Project Settings
  - Existing Code as Makefile Project
- CVS
- Git
- Install
- Maven

< Back | Next > | Cancel | Finish

Browse to MyTemperatureSensor, just select the top directory and OK.

Select Linux GCC as Toolchain and Finish

Window→Open Perspective→C/C++ and things should look like this:



File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Plug-in Development
PyDev
Debug
C/C++

Project Explorer

- pytest
- MyTemperatureSensor
  - Binaries
  - Includes
  - bin
  - obj
  - ClassFactory.cpp
  - main.cpp
  - MyTemperatureSensor.cpp
  - MyTemperatureSensor.h
  - MyTemperatureSensorClass.cpp
  - MyTemperatureSensorClass.h
  - MyTemperatureSensorStateMac
  - Makefile
  - Makefile.bck
  - MyTemperatureSensor.xmi

MyTemperatureSensor.h

```
/*----- PROTECTED REGION ID(MyTemperatureSensor.h) ENABLED START --


#ifndef MYTEMPERATURESENSOR_H
#define MYTEMPERATURESENSOR_H


#include <tango.h>

/*----- PROTECTED REGION END -----*/


/**
 *  MyTemperatureSensor class Description:
 *  MyTemperatureSensor
 */

namespace MyTemperatureSensor_ns
{
    /*----- PROTECTED REGION ID(MyTemperatureSensor::Additional Cla
```

- MYTEMPERATURESI
- tango.h
- MyTemperatureSen
  - MyTemperatureSe
    - attr_Temperatur
    - MyTemperature!
    - MyTemperature!
    - MyTemperature!
    - ~MyTemperature
    - delete_device() :
    - init_device() : vo
    - always_executec
    - read_attr_hardw

Problems | Tasks | Console | Properties

9 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Errors (9 items)				

/MyTemperatureSensor/MyTemperatureSensor.h

1. In the MyTemperatureSensor.cpp file

   a. Add initialization code in void MyTemperatureSensor::init_device()

Existing Code	New Code

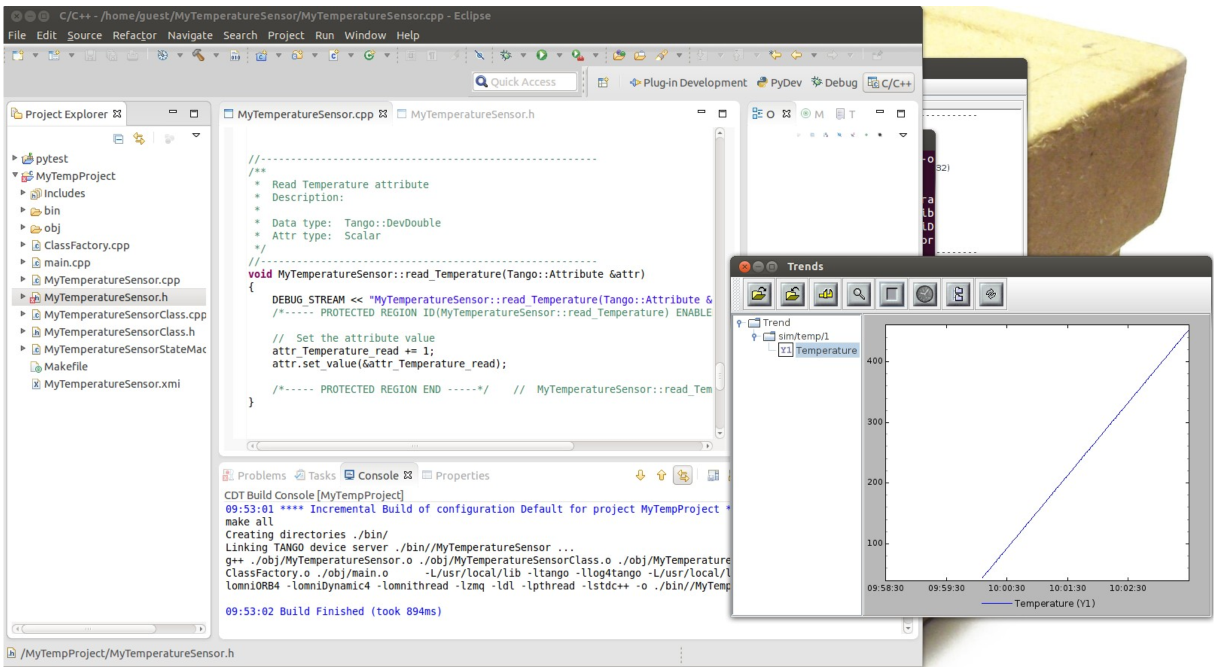
// Initialize device	// Initialize device
	set_state(Tango::ON);
	*attr_Temperature_read = 5;

   b. Set the variable to increase in void MyTemperatureSensor::read_Temperature(Tango::Attribute &attr)

Existing Code	New Code

// Set the attribute value
attr.set_value(attr_Temperature_read);
	*attr_Temperature_read += 1;
	
	// Set the attribute value
	attr.set_value(attr_Temperature_read);

2. Project→Build project or make in terminal
3. Restart device server
4. In Jive, monitor device to confirm that the temperature changes (increases).
5. With AtkMoni you can visualize the linear temperature increase.



```
//----------------------------------------------------------
/**
 *  Read Temperature attribute
 *  Description:
 *
 *  Data type:  Tango::DevDouble
 *  Attr type:  Scalar
 */
//----------------------------------------------------------
void MyTemperatureSensor::read_Temperature(Tango::Attribute &attr)
{
    DEBUG_STREAM << "MyTemperatureSensor::read_Temperature(Tango::Attribute &
    /*----- PROTECTED REGION ID(MyTemperatureSensor::read_Temperature) ENABLE

    //  Set the attribute value
    attr_Temperature_read += 1;
    attr.set_value(&attr_Temperature_read);

    /*----- PROTECTED REGION END -----*/    //  MyTemperatureSensor::read_Tem
}
```

CDT Build Console [MyTempProject]

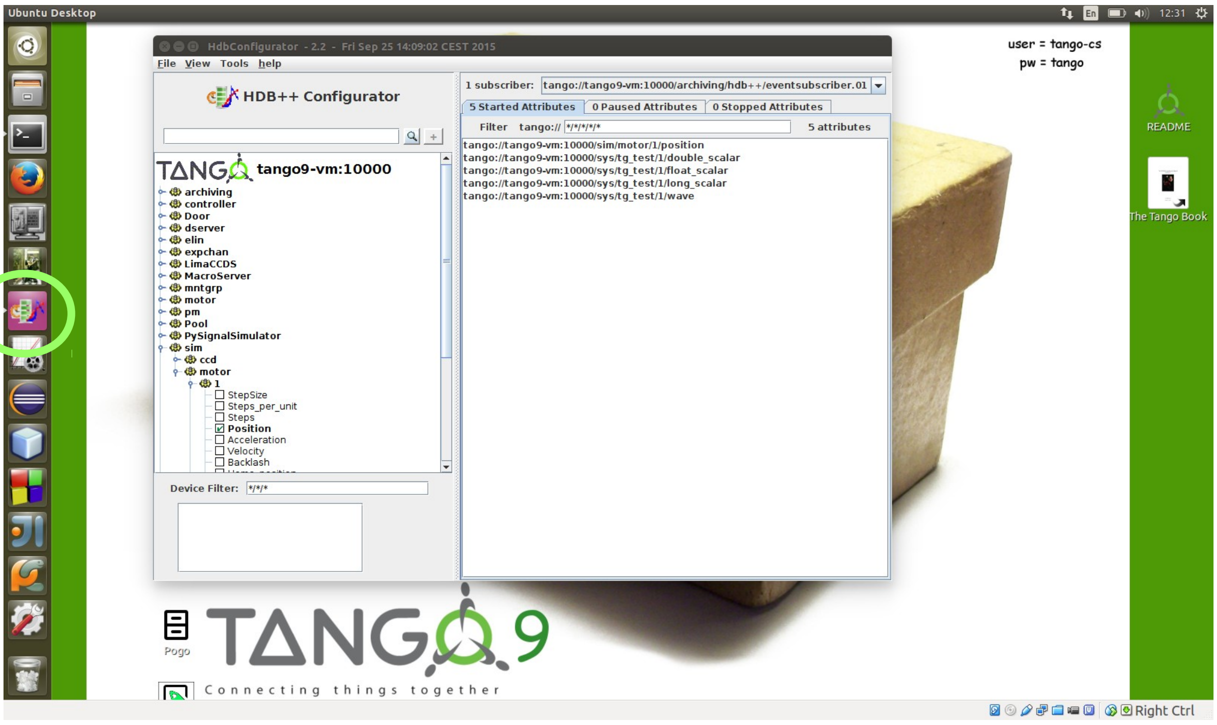
```
09:53:01 **** Incremental Build of configuration Default for project MyTempProject *
make all
Creating directories ./bin/
Linking TANGO device server ./bin//MyTemperatureSensor ...
g++ ./obj/MyTemperatureSensor.o ./obj/MyTemperatureSensorClass.o ./obj/MyTemperature
ClassFactory.o ./obj/main.o      -L/usr/local/lib -ltango -llog4tango -L/usr/local/l
lomniORB4 -lomniDynamic4 -lomnithread -lzmq -ldl -lpthread -lstdc++ -o ./bin//MyTemp

09:53:02 Build Finished (took 894ms)
```

Figure 12: Temperature is seen to increase

Historical DataBase: HDB++

TANGO offers 2 Historical Databases for archiving device attributes in a control system. The latest solution is called HDB++. It uses the events mechanism in device servers to trigger so-called “archive” events. These are gathered by archive event subscribers who listen for events and store them in the database. Two databases backends are implemented – a MySQL and a Cassandra one. The virtual machine has HDB++ installed for MySQL. It is pre-configured to read and stored 5 attributes. Two devices servers are running in the background and two graphical applications are provided via the launcher – hdb_config and hdb_viewer.

To start hdb_config click on the icon in the launcher. You will see a graphical window similar to this:



1 subscriber: tango://tango9-vm:10000/archiving/hdb++/eventsubscriber.01

Filter tango:// */*/*/* 5 attributes

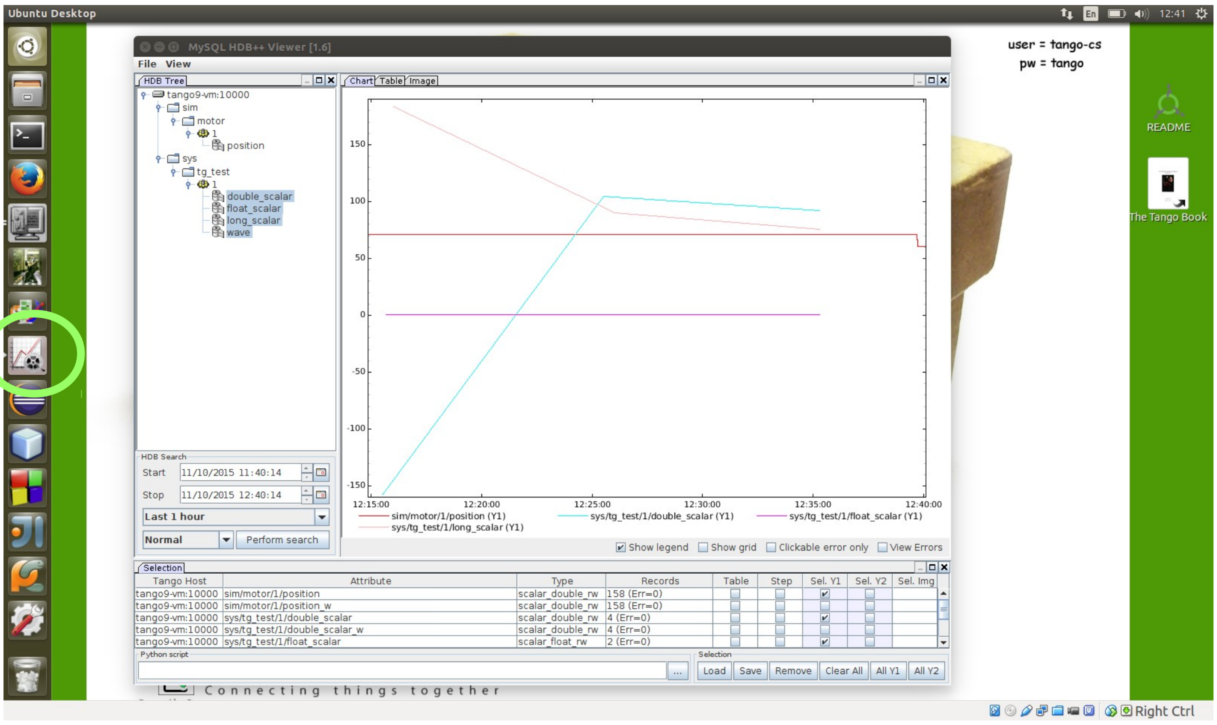
```
tango://tango9-vm:10000/sim/motor/1/position
tango://tango9-vm:10000/sys/tg_test/1/double_scalar
tango://tango9-vm:10000/sys/tg_test/1/float_scalar
tango://tango9-vm:10000/sys/tg_test/1/long_scalar
tango://tango9-vm:10000/sys/tg_test/1/wave
```

```
archiving
controller
Door
dserver
elin
expchan
LimaCCDS
MacroServer
mntgrp
motor
pm
Pool
PySignalSimulator
sim
  ccd
  motor
    1
      ☐ StepSize
      ☐ Steps_per_unit
      ☐ Steps
      ☑ Position
      ☐ Acceleration
      ☐ Velocity
      ☐ Backlash
```

Device Filter: */*/*

The TANGO device tree on the left can be used to select attributes and configure them for archiving. For events to work the device in question has to have polling configured or be capable of pushing events.

To view archived data from HDB++ start the hdb_viewer tool from the launcher to select which attributes to view and for which period:



HDB Search

Start 11/10/2015 11:40:14

Stop 11/10/2015 12:40:14

Last 1 hour

Normal — Perform search

[✓ Show legend] Show grid Clickable error only View Errors

Selection

Tango Host	Attribute	Type	Records	Table	Step	Sel. Y1	Sel. Y2	Sel. Img
tango9-vm:10000	sim/motor/1/position	scalar_double_rw	158 (Err=0)	☐	☐	✓	☐	
tango9-vm:10000	sim/motor/1/position_w	scalar_double_rw	158 (Err=0)	☐	☐	☐	☐	
tango9-vm:10000	sys/tg_test/1/double_scalar	scalar_double_rw	4 (Err=0)	☐	☐	✓	☐	
tango9-vm:10000	sys/tg_test/1/double_scalar_w	scalar_double_rw	4 (Err=0)	☐	☐	☐	☐	
tango9-vm:10000	sys/tg_test/1/float_scalar	scalar_float_rw	2 (Err=0)	☐	☐	✓	☐	

Tips and Tricks

The Tango Box image is configure with persistent storage on, changes you make are saved. When you shut down and restart, you continue where you left off, useful when you use it for development purposes. I you intend to use the Tango Box for demonstrations, you might want to start from the exact same image every time again, to avoid the notorious "demo effect".

The way to do this is to add this line to the scsi0 section of the .vmx file:

```
scsi0:0.mode = "independent-nonpersistent"
```