

Simulation in the Cloud

And a bit of Chaos engineering ...

Chaos Engineering wtf?

Build a Hypothesis around Steady State Behavior

Focus on the measurable output of a system, rather than internal attributes of the system. Measurements of that output over a short period of time constitute a proxy for the system's steady state. The overall system's throughput, error rates, latency percentiles, etc. could all be metrics of interest representing steady state behavior. By focusing on systemic behavior patterns during experiments, Chaos verifies that the system *does* work, rather than trying to validate *how* it works.

Vary Real-world Events

Chaos variables reflect real-world events. Prioritize events either by potential impact or estimated frequency. Consider events that correspond to hardware failures like servers dying, software failures like malformed responses, and non-failure events like a spike in traffic or a scaling event. Any event capable of disrupting steady state is a potential variable in a Chaos experiment.

Run Experiments in Production

Systems behave differently depending on environment and traffic patterns. Since the behavior of utilization can change at any time, sampling real traffic is the only way to reliably capture the request path. To guarantee both authenticity of the way in which the system is exercised and relevance to the current deployed system, Chaos strongly prefers to experiment directly on production traffic.

Automate Experiments to Run Continuously

Running experiments manually is labor-intensive and ultimately unsustainable. Automate experiments and run them continuously. Chaos Engineering builds automation into the system to drive both orchestration and analysis.

Minimize Blast Radius

Experimenting in production has the potential to cause unnecessary customer pain. While there must be an allowance for some short-term negative impact, it is the responsibility and obligation of the Chaos Engineer to ensure the fallout from experiments are minimized and contained.

How we Do Chaos Testing

We (ALBA) are migrating from a polling-based control system to an event-based approach.

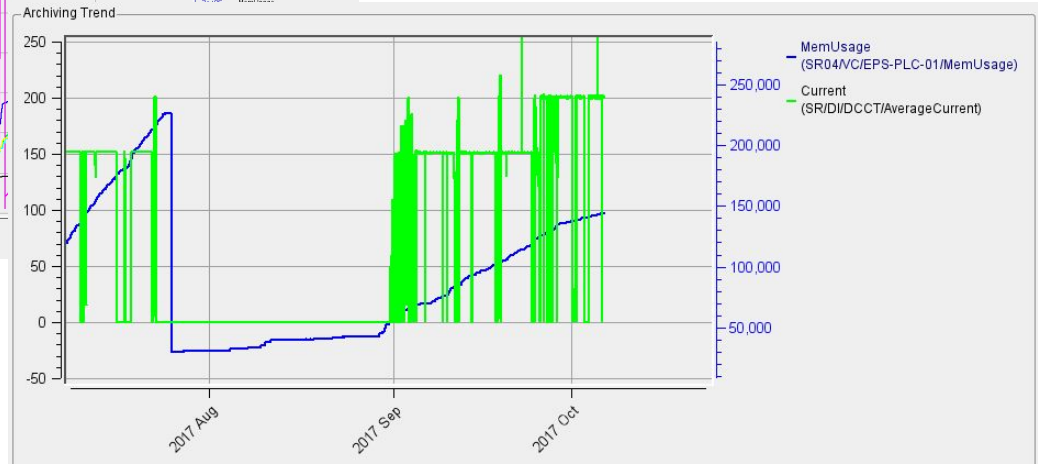
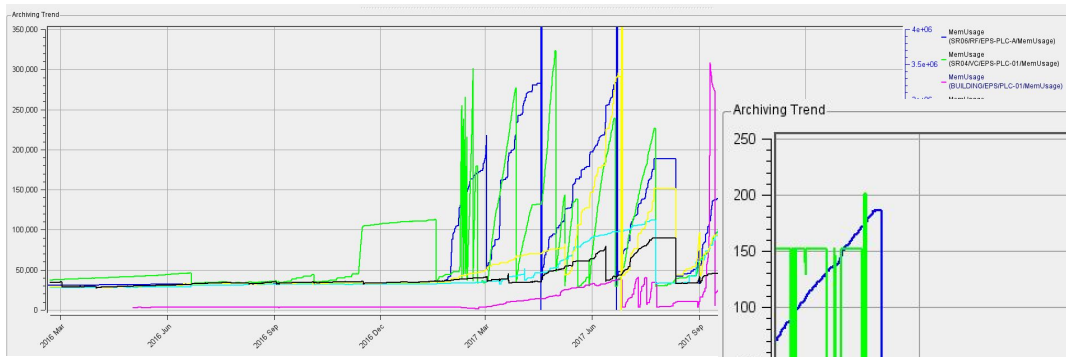
It affects cpu usage (threads), memory usage (buffers) and clients (exceptions, floods).

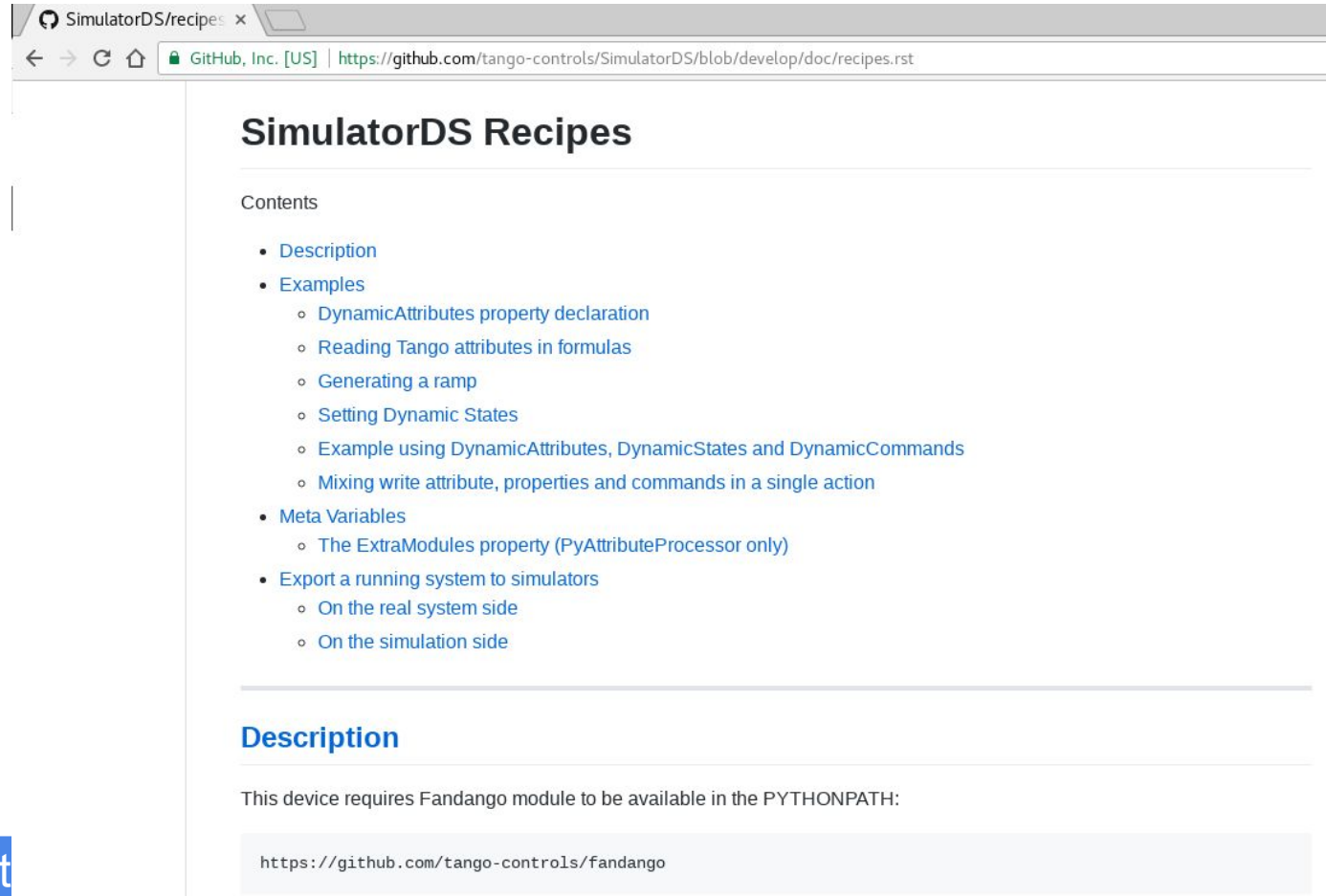
We try different event/polling configurations during machine's maintenance, or modify the behaviour of devices in a limited scope (few machines or only within a family). Then measuring the performance of GUI / Clients / Archiving.

Changes are running in few machines for 1-2 weeks before proceeding to upgrade (Canary Testing).

But if we have big problems to solve ... we don't have machine time for solving them. So we need a way to reproduce the bugs!

ProcessProfiler Device





SimulatorDS/recipe: x

← → ↻ 🏠 [GitHub, Inc. \[US\]](https://github.com) | <https://github.com/tango-controls/SimulatorDS/blob/develop/doc/recipes.rst>

SimulatorDS Recipes

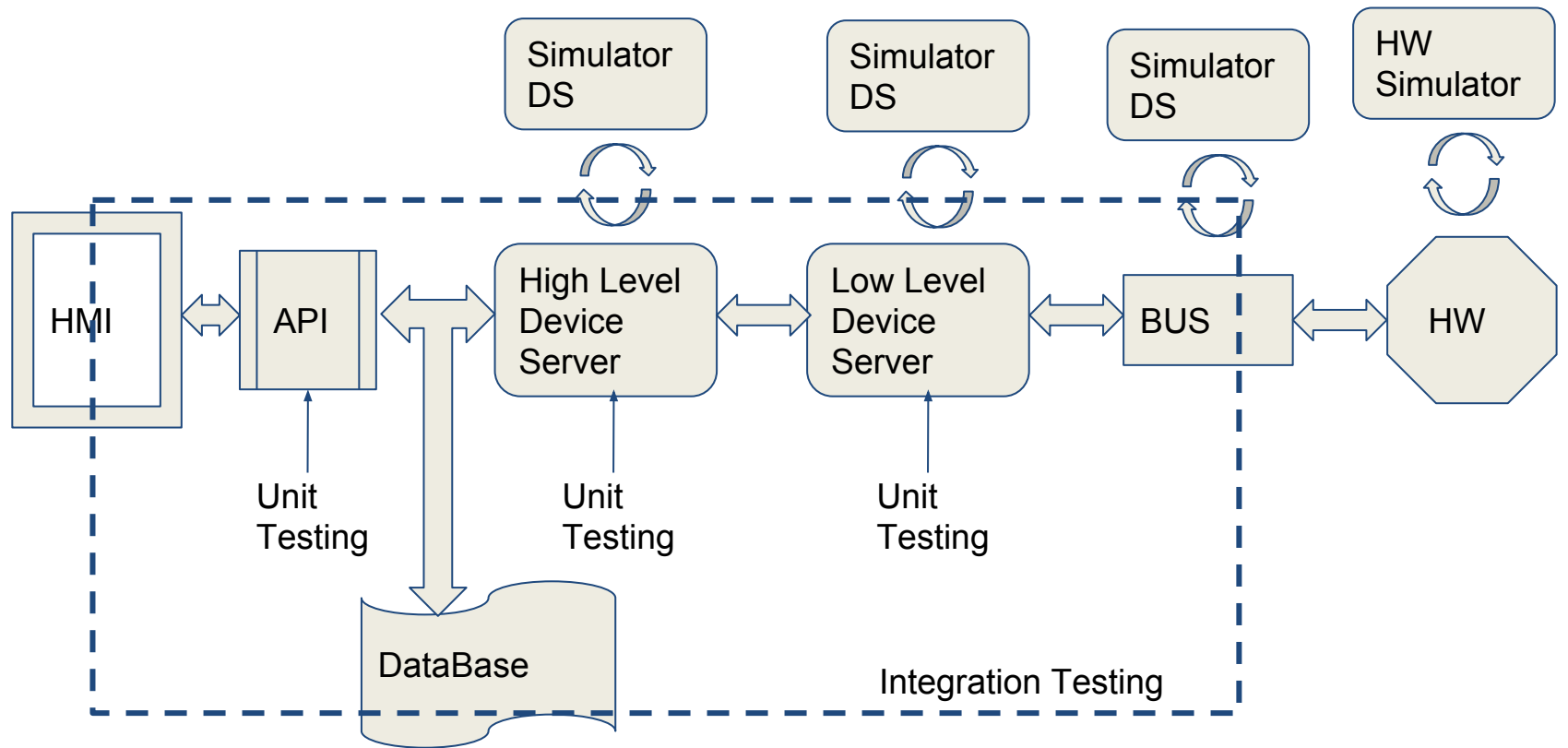
Contents

- [Description](#)
- [Examples](#)
 - [DynamicAttributes property declaration](#)
 - [Reading Tango attributes in formulas](#)
 - [Generating a ramp](#)
 - [Setting Dynamic States](#)
 - [Example using DynamicAttributes, DynamicStates and DynamicCommands](#)
 - [Mixing write attribute, properties and commands in a single action](#)
- [Meta Variables](#)
 - [The ExtraModules property \(PyAttributeProcessor only\)](#)
- [Export a running system to simulators](#)
 - [On the real system side](#)
 - [On the simulation side](#)

Description

This device requires Fandango module to be available in the PYTHONPATH:

```
https://github.com/tango-controls/fandango
```



SimulatorDS

One of the paradigms presented by SimulatorDS is that the code no longer is stored in a disk or a machine.

The code is loaded from databases, and is mutable on runtime. This capability is exploited in other several PyTango projects:

PyStateComposer

PyAttributeProcessor

CopyCatDS

WorkerDS (remote script executor)

PANIC (ALBA's Alarm System)

Device properties [test/sr/di]

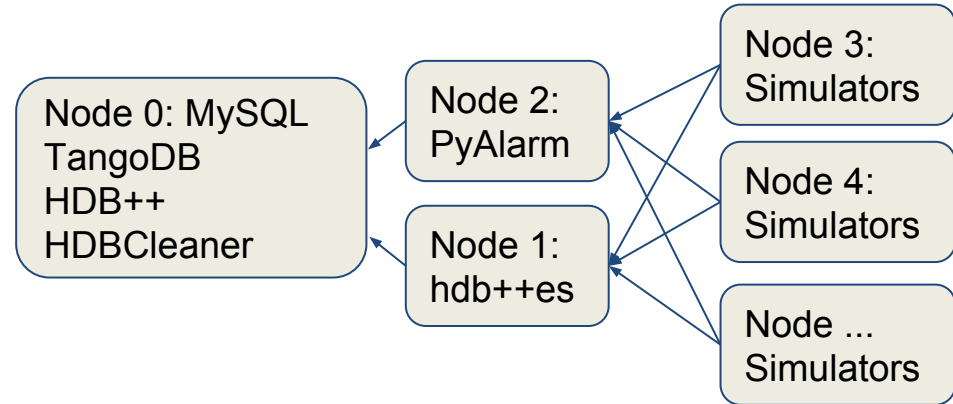
Property name	Value
CheckDependencies	True
DynamicAttributes	CRef=float(HDB.get_attribute_values('sr/di/dcct/current',str2time('2017-03-01'),'+300')[N]) PRef=float(HDB.get_attribute_values('sr/vc/all/pressure',str2time('2017-03-01'),'+300')[N]) Noise = sin(PI*t) Pressure = PRef+ 1e-7*Noise Current = CRef+ 2*Noise
DynamicStates	ALARM = Pressure > 1e-8 MOVING = 0 < Current < 20 ON = Current > = 20 OFF = 1
ExtraModules	PyTangoArchiving.Reader as HDB lifetime_lib
KeepAttributes	True
KeepTime	500
LoadFromFile	/control/sr_di_calc.py
LogLevel	WARNING
UseEvents	false

Chaos Testing on AWS

Andy proposed to create a testing platform on AWS

We developed scripts to:

- export a control system to simple .csv files
- simplify aws-cli usage
- start/create/list instances
- extract public DNS
- create and configure devices
- modify device setup easily



EC2 Dashboard

Events

Tags

Reports

Limits

INSTANCES

Instances

Spot Requests

Reserved Instances

Dedicated Hosts

IMAGES

AMIs

Bundle Tasks

Launch Instance Connect Actions

Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm
<input checked="" type="checkbox"/>	tango-chaos-0	i-050c3153014ffe582	t2.large	eu-central-1b	● running	✔ 2/2 checks...	None
<input type="checkbox"/>	tango-chaos-1	i-0c9500d0d321cfbdd	t2.micro	eu-central-1b	● running	✔ 2/2 checks...	None
<input type="checkbox"/>	tango-chaos-2	i-0cc76007b2d60af11	t2.micro	eu-central-1b	● running	✔ 2/2 checks...	None
<input type="checkbox"/>	tango-chaos-3	i-0f8898b6d94698382	t2.micro	eu-central-1b	● running	✔ 2/2 checks...	None
<input type="checkbox"/>	tango-chaos-4	i-05460cace6b0bb315	t2.micro	eu-central-1b	● running	✔ 2/2 checks...	None
<input type="checkbox"/>	tango-test	i-0e22c479940c9cece	t2.micro	eu-central-1b	● stopped		None

fandangoing

In bash or python:

```
$ fandango add_new_device SimulatorDS/testrw SimulatorDS test/tango/rw
$ fandango put_device_property test/tango/rw DynamicAttributes \
  'A=VAR("A",WRITE=True,default=0)' \
  'B=VAR("B",WRITE=True,default=0)'

$ tango_servers tango-chaos-0 start SimulatorDS/testrw

$ for attr in $(fandango find_attributes "test/tango/rw/*"); do
> echo "$attr : $(fandango read_attribute $attr)"
> done
test/tango/rw/A : 0.0
test/tango/rw/B : 0.0
test/tango/rw/MemUsage : 77904.0
test/tango/rw/State : ON
test/tango/rw/Status : The device is in ON state.
```

.csv? human readable

```
$ tango2csv test/tango/rw rw.csv
```

```
$ csv2tango rw.csv
```

```
$ tango_servers stop test/tango/rw
```

```
$ tango_servers start test/tango/rw
```

```
$ ipython
```

```
: from PyTango import DeviceProxy
```

```
: dp = DeviceProxy ('test/tango/rw')
```

```
: dp.write_attributes(
```

```
    [('a',2),('b',20)])
```

```
    DevFailed: DevFailed[
```

```
    DevError[
```

```
        desc = Set value for attribute B is above the maximum authorized (at least element 0)
```

```
        origin = WAttribute::check_written_value()
```

```
        reason = API_WAttrOutsideLimit
```

```
        severity = ERR]
```

```
: [v.value for v in dp.read_attributes(['a','b'])]
```

```
Out:: [2.0, 0.0]
```

	A	B	C	D	E
1	server	class	device	property	value
2	<u>SimulatorDS/testrw</u>	<u>SimulatorDS</u>	<u>test/tango/rw</u>	<u>DynamicAttributes</u>	A=VAR("A",WRITE=True,default=0)
3					B=VAR("B",WRITE=True,default=0)
4				<u>B.min_value</u>	0
5					

	A	B	C	D	E
1	server	class	device	property	value
2	<u>SimulatorDS/testrw</u>	<u>SimulatorDS</u>	<u>test/tango/rw</u>	<u>DynamicAttributes</u>	A=VAR("A",WRITE=True,default=0)
3					B=VAR("B",WRITE=True,default=0)
4				<u>B.min_value</u>	0
5				<u>B.max_value</u>	16
6					

A	B	C	D	E	
host	server	class	device	property	value
Tango-chaos-4	<u>DynamicDS/elinac-beam</u>	<u>SimulatorDS</u>	<u>elin/beam/run</u>	<u>DynamicAttributes</u>	Cleaning = DevDouble(VAR('Cleaning', default=True * (1+2*sin((int(PROPERTY("OFFSE DeflectionDC = DevDouble(VAR('DeflectionDC', default=0.0 * (1+2*sin((int(PROPERTY("I Delay = DevDouble(VAR('Delay', default=6.93217931773e-310 * (1+2*sin((int(PROPERTY("I GridV = DevDouble(VAR('GridV', default=-72.0 * (1+2*sin((int(PROPERTY("OFFSET")))+t PulseL = DevDouble(VAR('PulseL', default=2.1 * (1+2*sin((int(PROPERTY("OFFSET")))+t PulseType = DevDouble(VAR('PulseType', default=SHORT * (1+2*sin((int(PROPERTY("C PulseV = DevDouble(VAR('PulseV', default=15.6 * (1+2*sin((int(PROPERTY("OFFSET"))
				<u>DynamicCommands</u>	Off = DevString('elin/beam/run/Off') On = DevString('elin/beam/run/On') Reset = DevString('elin/beam/run/Reset')
				<u>DynamicStates</u>	MOVING=t%randint(15,30)>randint(1,15) ALARM=t%randint(1,30)<randint(1,6) ON=1
				<u>LoadFromFile</u>	/home/srubio/CO/vacca/examples/elinac/LinacGun_attributes.txt
				<u>OFFSET</u>	
Tango-chaos-5	<u>DynamicDS/elinac-cool</u>	<u>SimulatorDS</u>	<u>elin/cool/1</u>	<u>DynamicAttributes</u>	BuncherTemp = DevDouble(VAR('BuncherTemp', default=27.1 * (1+2*sin((int(PROPERTY PBuncherTemp = DevDouble(VAR('PBuncherTemp', default=26.1 * (1+2*sin((int(PROPEF SectionTemp = DevDouble(VAR('SectionTemp', default=28.1 * (1+2*sin((int(PROPERTY(WaterTemp = DevDouble(VAR('WaterTemp', default=25.1 * (1+2*sin((int(PROPERTY("OF
				<u>DynamicCommands</u>	Reset = DevString('elin/cool/1/Reset')
				<u>DynamicStates</u>	MOVING=t%randint(15,30)>randint(1,15) ALARM=t%randint(1,30)<randint(1,6) ON=1
				<u>LoadFromFile</u>	/home/srubio/CO/vacca/examples/elinac/LinacCooling_attributes.txt
				<u>OFFSET</u>	
Tango-chaos-5	<u>DynamicDS/elinac-cool</u>	<u>SimulatorDS</u>	<u>elin/cool/bun-temps</u>	<u>DynamicAttributes</u>	Current = DevDouble(VAR('Current', default=223.0 * (1+2*sin((int(PROPERTY("OFFSET" Frequency = DevDouble(VAR('Frequency', default=3140.0 * (1+2*sin((int(PROPERTY("O Interlocks = DevVarDoubleArray(VAR('Interlocks', default=[0 * (1+2*sin((int(PROPERTY(Position = DevDouble(VAR('Position', default=54.5 * (1+2*sin((int(PROPERTY("OFFSE

PyAlarm/Clock	PyAlarm	test/panic/ck02	<u>AlarmList</u>	CK02:False and not CK02	
			<u>AlarmSeverities</u>	CK02:DEBUG	
			<u>AlarmThreshold</u>		1
	test/panic/ck05	<u>AlarmList</u>	CK05: False and not CK05		
		<u>AlarmSeverities</u>	CK05:DEBUG		
		<u>AlarmThreshold</u>		2	
		<u>PollingPeriod</u>		0.125	
	test/panic/ck1	<u>AlarmList</u>	CK1: not CK1		
		<u>AlarmSeverities</u>	CK1:DEBUG		
		<u>AlarmThreshold</u>		5	
	test/panic/ck2	<u>AlarmList</u>	CK2: NOW()%2 < 1		
		<u>AlarmSeverities</u>	CK2:DEBUG		
		<u>AlarmThreshold</u>		1	
	test/panic/ck5	<u>AlarmList</u>	CK5: NOW()%5<2.5		
		<u>AlarmSeverities</u>	CK5:DEBUG		
<u>AlarmThreshold</u>			1		
test/panic/ck10	<u>AlarmList</u>	CK10:NOW()%10<5			
	<u>AlarmSeverities</u>	CK10:DEBUG			
	<u>AlarmThreshold</u>		1		
PyAlarm/Group	PyAlarm	test/panic/group	<u>AlarmList</u>	GROUP_0: GROUP(test/panic/ck*/ck5 ck10) GROUP_1: GROUP(CK5,CK10) GROUP_OR: GROUP(CK5,CK10,x>=1) GROUP_AND: test/panic/ck2/ck2 and test/panic/ck5/ck5 GROUP_ALL: GROUP(GROUP_0,GROUP_1,GROUP_2,GROUP_AND)	
			<u>AlarmSeverities</u>	GROUP_1: ALARM GROUP_OR: ALARM GROUP_AND: ALARM GROUP_ALL:ALARM	
			<u>AutoReset</u>		0.0001
			<u>PollingPeriod</u>		0.5
			<u>AlarmThreshold</u>		1

Use case: OSS collaboration

A bug is found, but it is only reproduceable on a given system setup

Send devices and attributes configurations in csv files

Send the distribution of devices/servers in hosts in another file

Reproduce a reported bug in the cloud, debug, terminate servers afterwards

For more info see TUDPL01 on Tuesday ...