PANIC, The ALBA Alarm System

Package for Alarms and Notification of Incidences from Controls

Sergi Rubio Manrique, ALBA Controls Section

What an Alarm System should do:

- Verification of a set of conditions.

- Notification.

- Keep a log of what happened.

- Take automatic actions?

- Tools for configuration/visualization.

Alarm Life Cycle

- Condition: CIRCE_PRESSURE:BL24/VC/VGCT-01/P1>3e-5

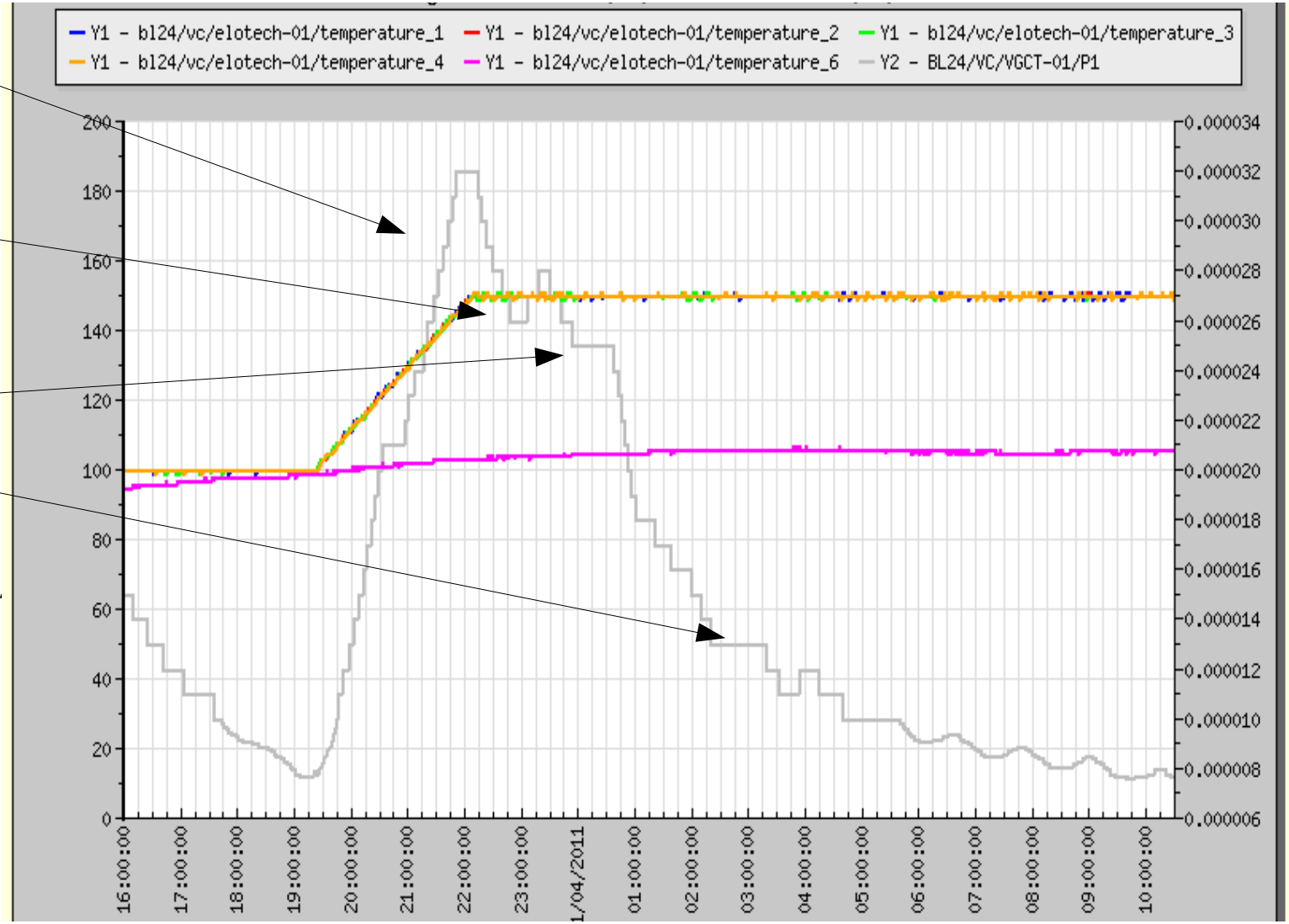
- Incidence
- Notification
- Logging

- Recovery
- Reminder

- Acknowledge

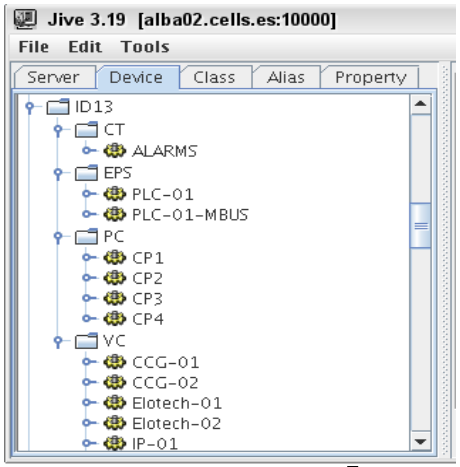
- Reset
- Logging

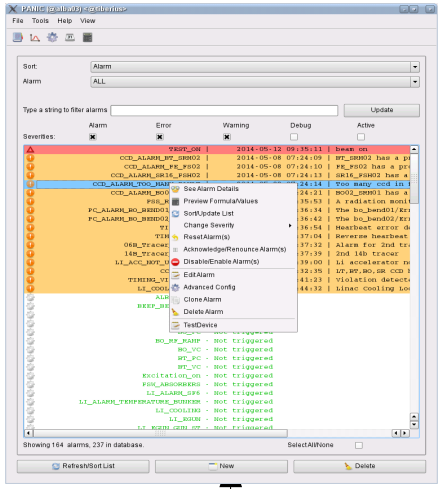
- Further Actions

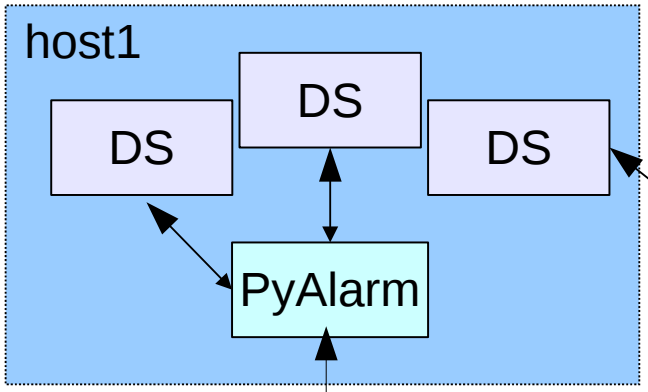


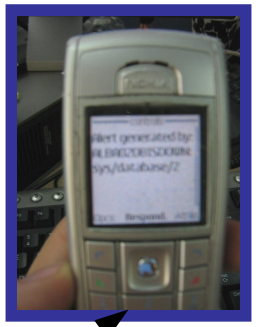
PANIC, The Alba Alarm System

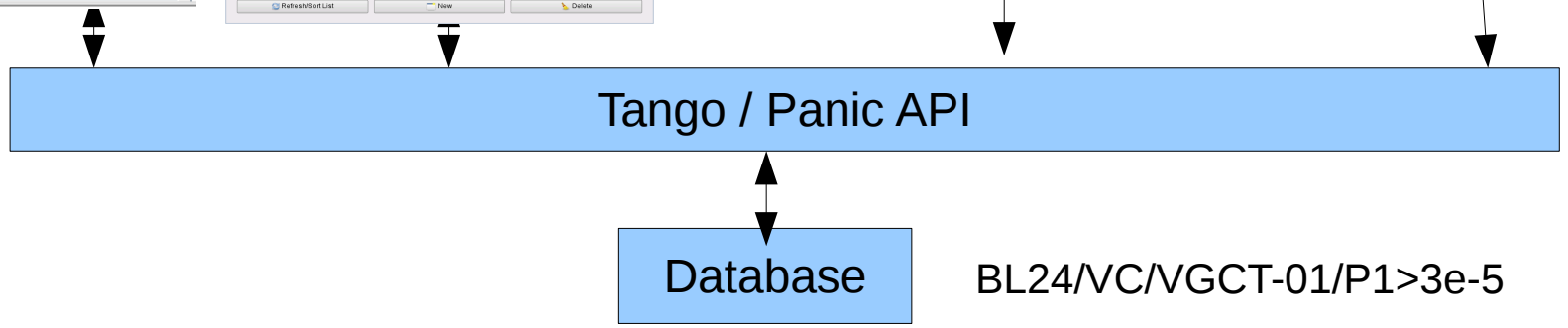
- Distributed in PyAlarm Device Servers, each managing a collection of **alarm formulas**.
- **Each Alarm is unique** in the system and can be managed by only 1 PyAlarm device.
- Each PyAlarm device performs locally **both Logging and Notification (email/SMS)**
- **Additional actions** are passed **to external devices** (SnapArchiver, PopupDevice)
- Configuration is stored in the **Tango Database**.
- **Alarm logging** in the Tango **Snapshoting database**.
- The Panic API provides an **homogeneous view** of the system for devices and GUI.











BL24/VC/VGCT-01/P1>3e-5

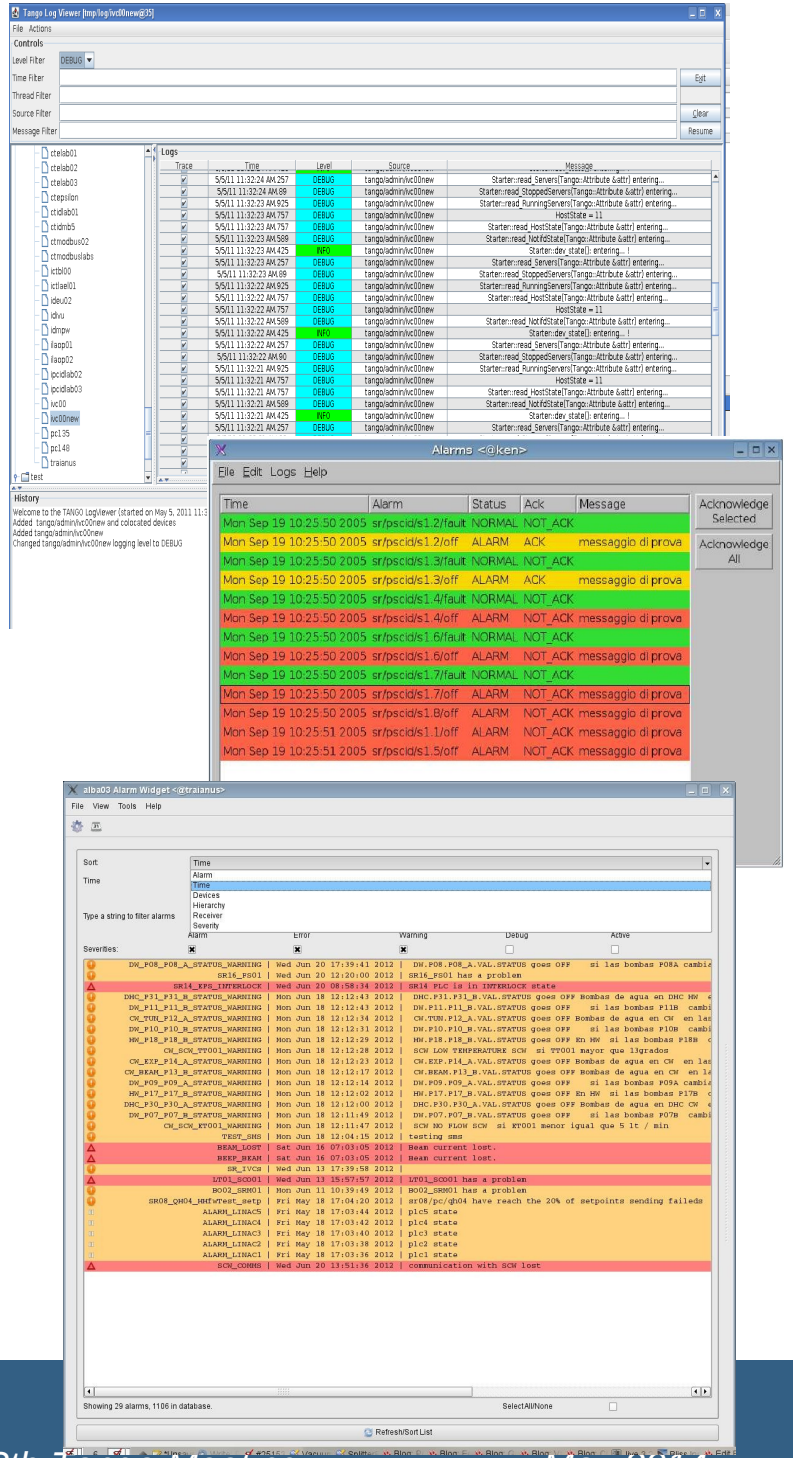
Yet another Alarm System??

The core of Panic is the **PyAlarm device**, developed in 2007 as an SMS Notification Device for **Tango Alarm System (Elettra's)**

Both Panic and Elettra Alarm Systems evaluate evaluate **formulas of Tango attributes** and return boolean results. Both are capable of **triggering commands on other devices.**

Unlike Tango Alarm System (as it was in 2007), PANIC is **distributed and doesn't require other database than Tango**. It was developed to use no-db device servers in isolated machines

Like Soleil's alarm system it can be configured to trigger **alarms based on Attribute qualities**, and do not require any client as the devices operate stand-alone.

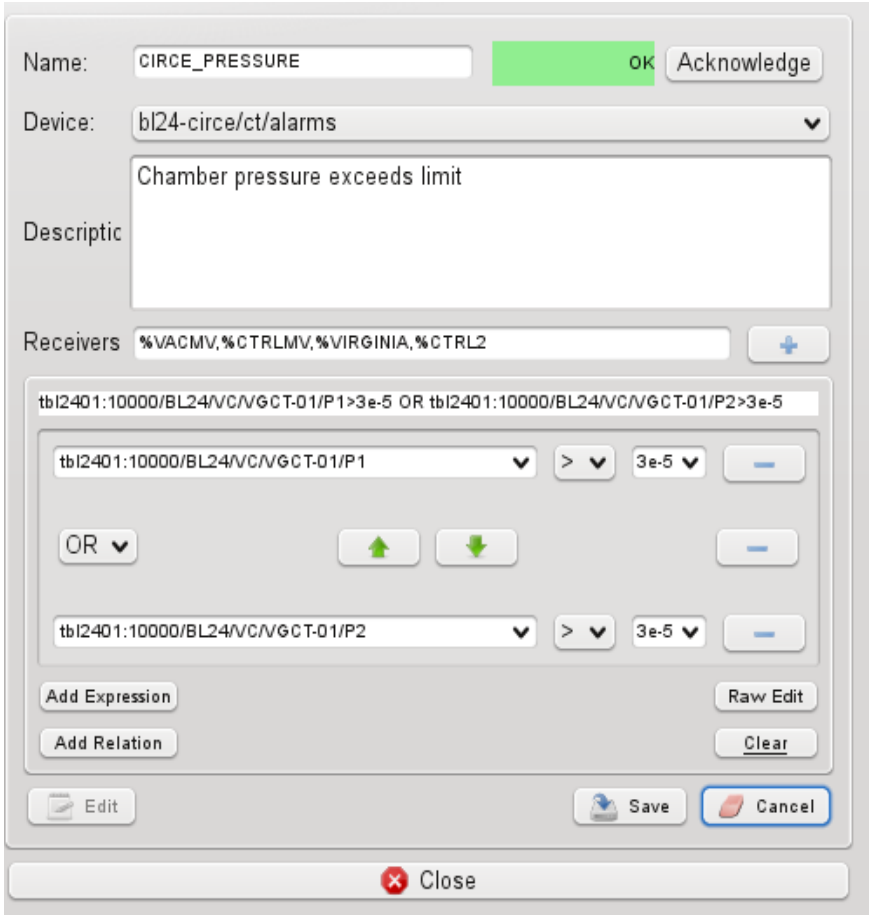


Declaring Alarms

Each Alarm is defined by:

TAG,
Formula,
Description,
Receivers,
Severity,
{Configuration})

Those are stored using **device properties** of PyAlarm devices.



Alarm formulas:

CIRCE_PRESSURE: **tbl2401:10000**/BL24/VC/VGCT-01/P1>3e-5

CIRCE_LOST: BL24/VGCT-01/State == **UNKNOWN**

CIRCE_TEMP: BL24/EPS/PLC-01/T1**.quality** == **ATTR_ALARM**

CIRCE_VALVE: FE24/VC/PNV-01/State**.delta**!=0

Declaring Alarms with Regular Expressions

any([t>85 for t in FIND(ID13/EPS/PLC-01/TTAP*****_VAL)])

any([min(t)<50 and 70<max(t)<1000 for t in **[FIND(**ID13/VC/Elotech*/Temperature_[0-9])]])

any([t==ATTR_ALARM for t in **FIND(**ID13/VC/Elotech*/Temperature**_[0-9].quality**)])

GROUP(BL24/CT/ALARMS/CIRCE_*)

Full Tango Attribute URL:

[tango:host/][device/]Attribute[.value/quality/delta/time/exception/all]g

Yet another fandango.eval format ...

fandango.dynamic.DynamicDS
(PyPLC, PySignalSimulator, PyAttributeProcessor, PyStateComposer, ...)

```
MaxP=DevBoolean(XATTR('tbl24:10000/BL24/VC/VGCT-01/P1')>MaxV)
CCGs=DevVarStringArray([d+'/P'+a for a in '12' for d in DEVICES if 'VG' in d])
AlarmP=DevBoolean(any(Q(c) for c in CCGs)
AnyP=DevBoolean(MaxP or AlarmP)
```

It allows to use commands and attributes **within a Tango device.**
Parsed/executed by Device commands, tied to **DevImpl** class methods
Variables **isolated** between instances**, kept only when needed**

fandango.tango.TangoEval *(PyAlarm , Panic UI)*

```
MaxP=BL24/VC/VGCT-01/P1>3e-5
AlarmP=any(q in (ATTR_ALARM,) for q in FIND(BL/VC/VG-*/P*.quality))
AnyP=MaxP or AlarmP
```

Same eval object within **devices and clients**
Simplified syntax, enabling **wildcards** and **delta/quality/time** comparison
Keeps cache and values for all evaluated objects, **shared between instances**
Both approaches use lazy **polling by expiration date (**CachedAttributeProxy**)**

Database dependency

PyAlarm uses TANGO as Alarms Configuration Database, storing the Alarm configurations using Device and Class properties (Phonebook, SMSConfig):

- Pro: It just needs the TANGO database (or a no-db file) to work.
- Cons: It is filling the Tango database with a lot of ugly information.
- Merging with Elettra's database has been delayed for years.

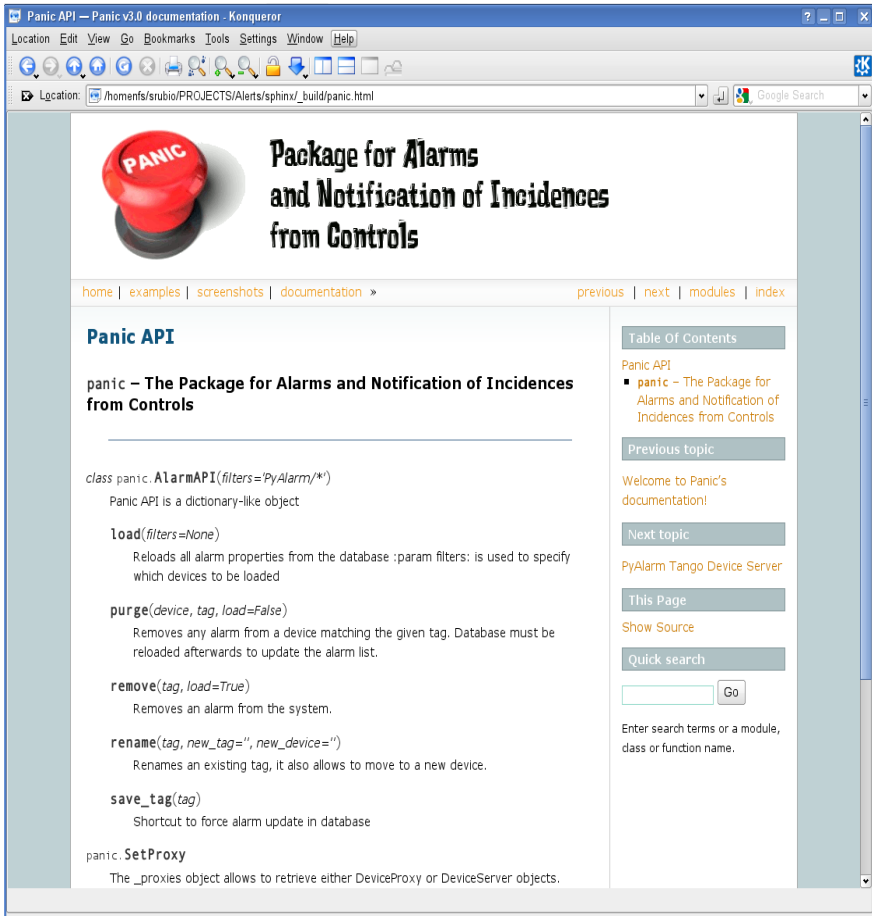
Thanks to Panic API the device server and the GUI are completely independent from the database. It will allow us to adapt our servers to some tools developed by Elettra and viceversa (when needed).

AlarmsList	CIRCE_PRESSURE:any([pressure>2e-5 for pressure in [BL24-CIRCE/VC/VGCT-01/P2,BL24-CIRCE/VC/VGCT-02/P2]]) CIRCE_TEMPERATURE1:any(BL24-CIRCE/VC/Elotech-01/Temperature_All[i]>t for i,t in enumerate([190,190,160,160,130,130,13 CIRCE_TEMPERATURE2:any(BL24-CIRCE/VC/Elotech-02/Temperature_All[i]>t for i,t in enumerate([130,130,130,130,130,130,16 CIRCE_TEMPERATURE3:any(BL24-CIRCE/VC/Elotech-03/Temperature_All[i]>t for i,t in enumerate([170,200,120,120,140,170])) CIRCE_TEMPERATURES:CIRCE_TEMPERATURE1 or CIRCE_TEMPERATURE2 or CIRCE_TEMPERATURE3 CIRCE_STOP:BL24-CIRCE/VC/Elotech-01/State != ON CIRCE_LOST:any([state==UNKNOWN for state in [BL24-CIRCE/VC/VGCT-01/State,BL24-CIRCE/VC/Elotech-01/State]]) CIRCE_STARTED:max(BL24-CIRCE/VC/Elotech-01/Temperature_All) > 35
LogFile	/tmp/alarm_FrontEnds-CT-Alarms.log

The Panic Module

Panic contains the python AlarmAPI for managing the PyAlarm device servers from a client application or a python shell. The panic module is part of the Panic bliss package.

```
import panic
alarms = panic.api()
```



Browsing existing alarms

The AlarmAPI is a dictionary-like object containing Alarm objects for each registered Alarm tag. In addition the AlarmAPI.get method allows caseless search by tag, device, attribute or receiver:

```
alarms.get(self, tag='', device='', attribute='', receiver='')
```

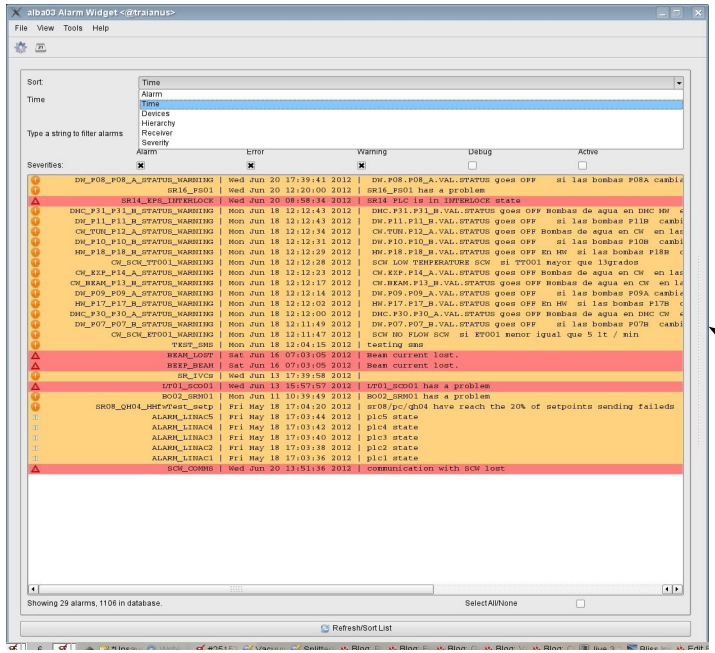
```
alarms.get(device='boreas')
Out[232]:
[Alarm(BL29-BOREAS_STOP:The BakeOut controller has been stop),
 Alarm(BL29-BOREAS_PRESSURE_1:),
 Alarm(BL29-BOREAS_PRESSURE_2:),
 Alarm(BL29-BOREAS_START: BL29-BOREAS bakeout started
 …]

alarms.get(receiver='eshraq')
Out[234]:
[Alarm(RF_LOST_EUROTHERM:),
 Alarm(OVEN_COMMS_FAILED:Oven temperatures not updated in the last
5 minutes),
 Alarm(RF_PRESSURE:The pressure in the cavity exceeds Range),
 Alarm(OVEN_TEMPERATURE:The Temperature of the Oven exceeds
Range),
 Alarm(RF_EUROTHERM:),
 Alarm(RF_LOST_MKS:),
 Alarm(RF_TEMPERATURE_MAX2:),
 ...]
alarms['RF_LOST_MKS'].receivers
Out[237]: '%SRUBIO,%ESHRAQ,%VACUUM,%LOTHAR,%JNAVARRO'
```

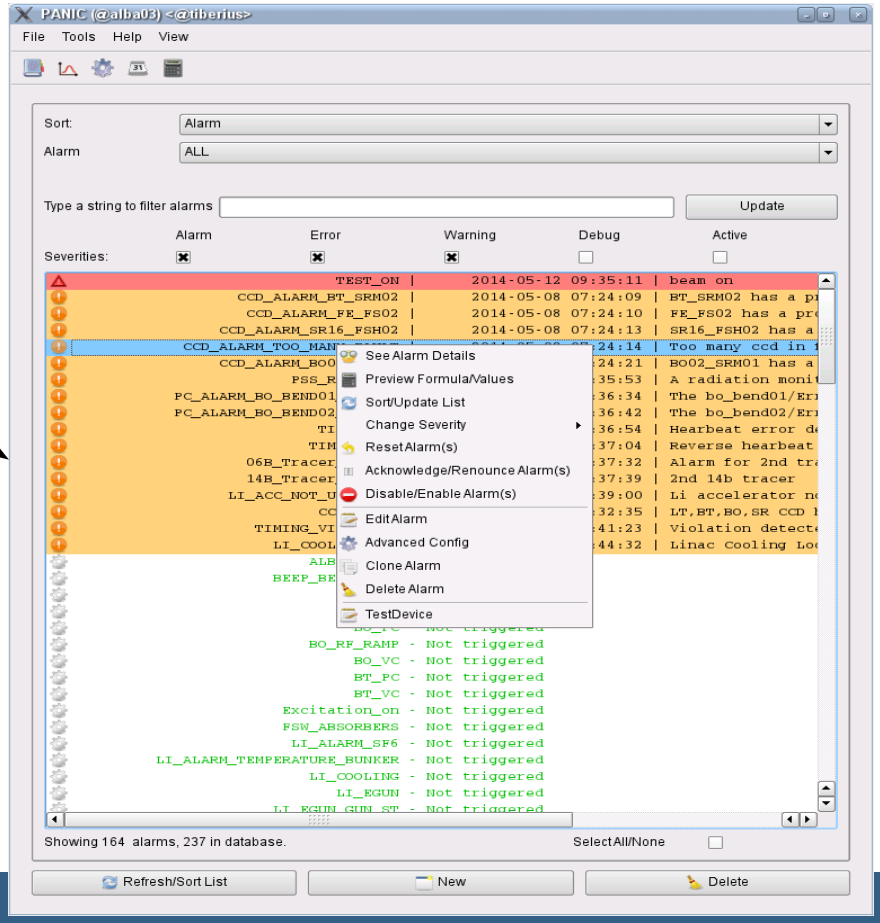
Panic User Interface

The Panic tool shows the list of active or declared alarms. It provides several filters to search alarms: by state (active/inactive), severity, subsystem, receiver or historic values.

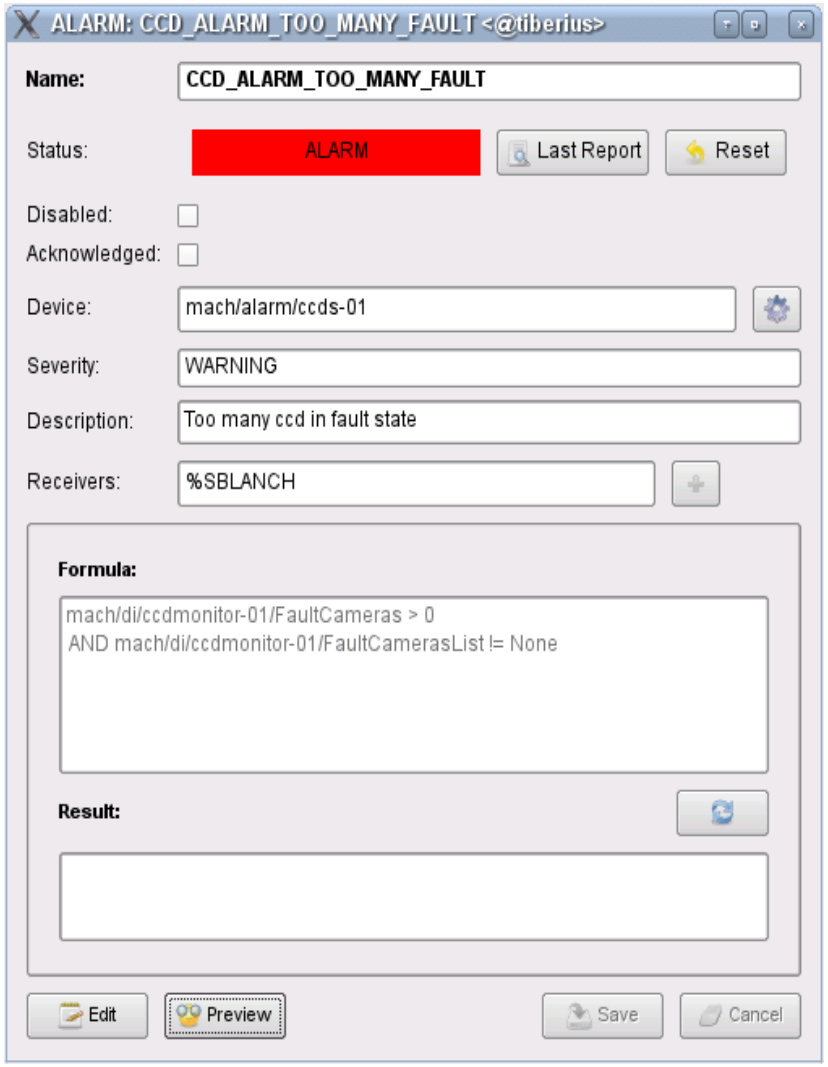
A text search is also provided that allow to locate alarms by any of the attributes used in formula or words used in description.

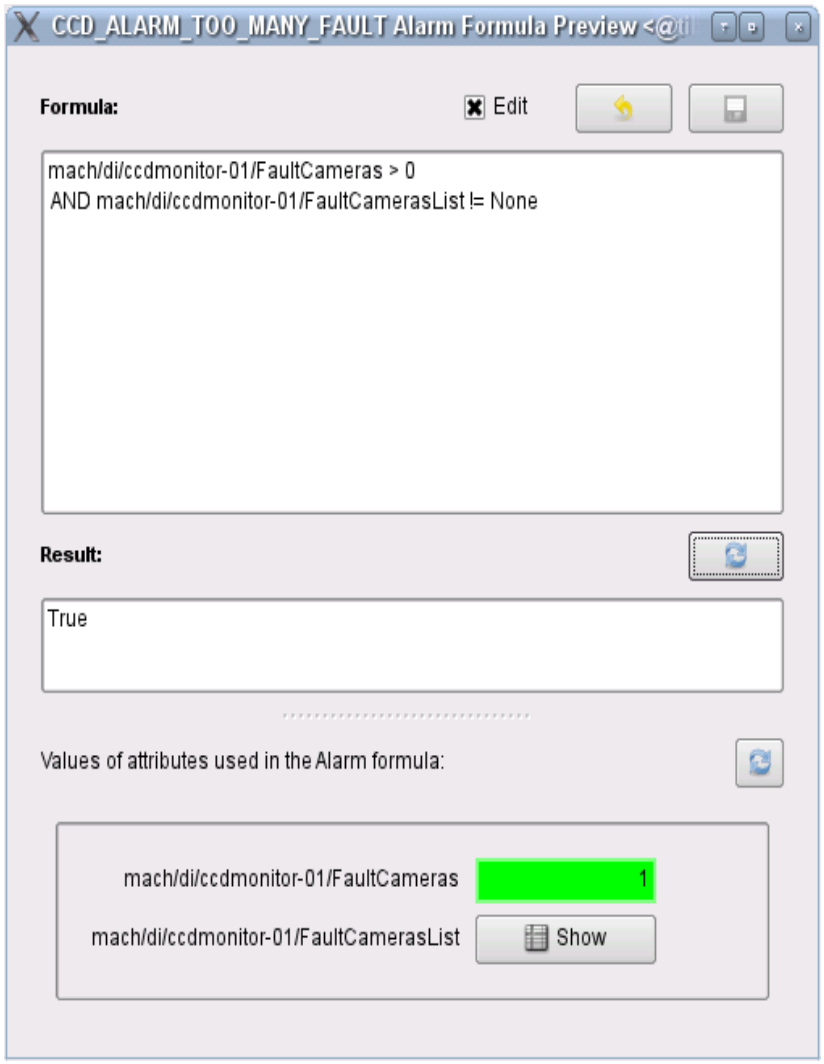
For each alarm the menu allows to Reset the alarm or show the attribute values that triggered it.





UI Editor and Preview



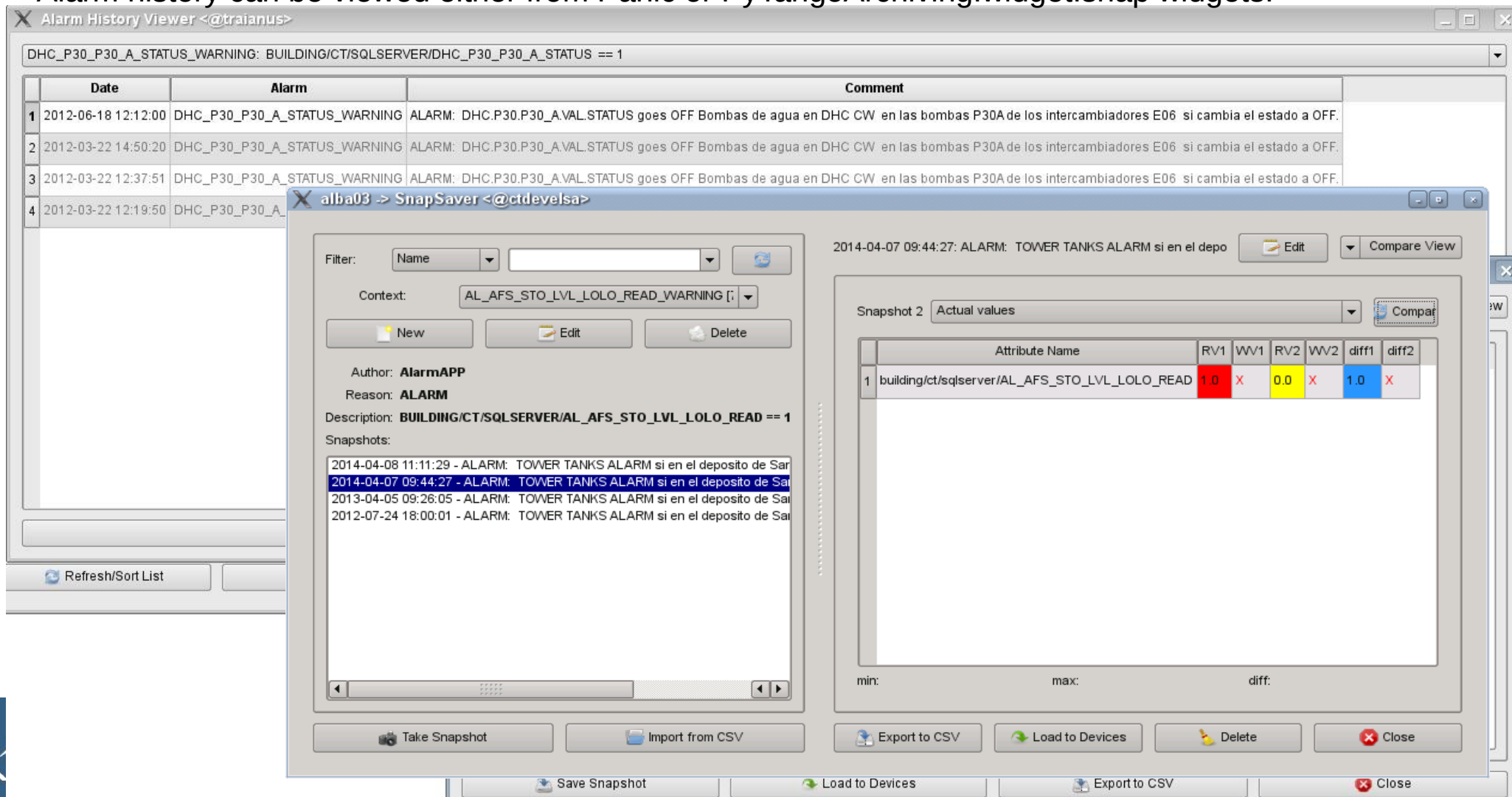


Record/View Alarm History using Snaps

If alarm history is enabled, (**SNAP Receiver or UseSnaps+CreateNewContext** property) then attribute values will be recorded every time that the alarm is triggered.

The alarm will create a context in the Tango Snapshoting database with all the attributes that appear in the formula. It can be modified later to include additional attributes.

Alarm history can be viewed either from Panic or PyTangoArchiving.widget.snap widgets.



Alarm Life Cycle

- Condition: CIRCE_PRESSURE:BL24/VC/VGCT-01/P1>3e-5

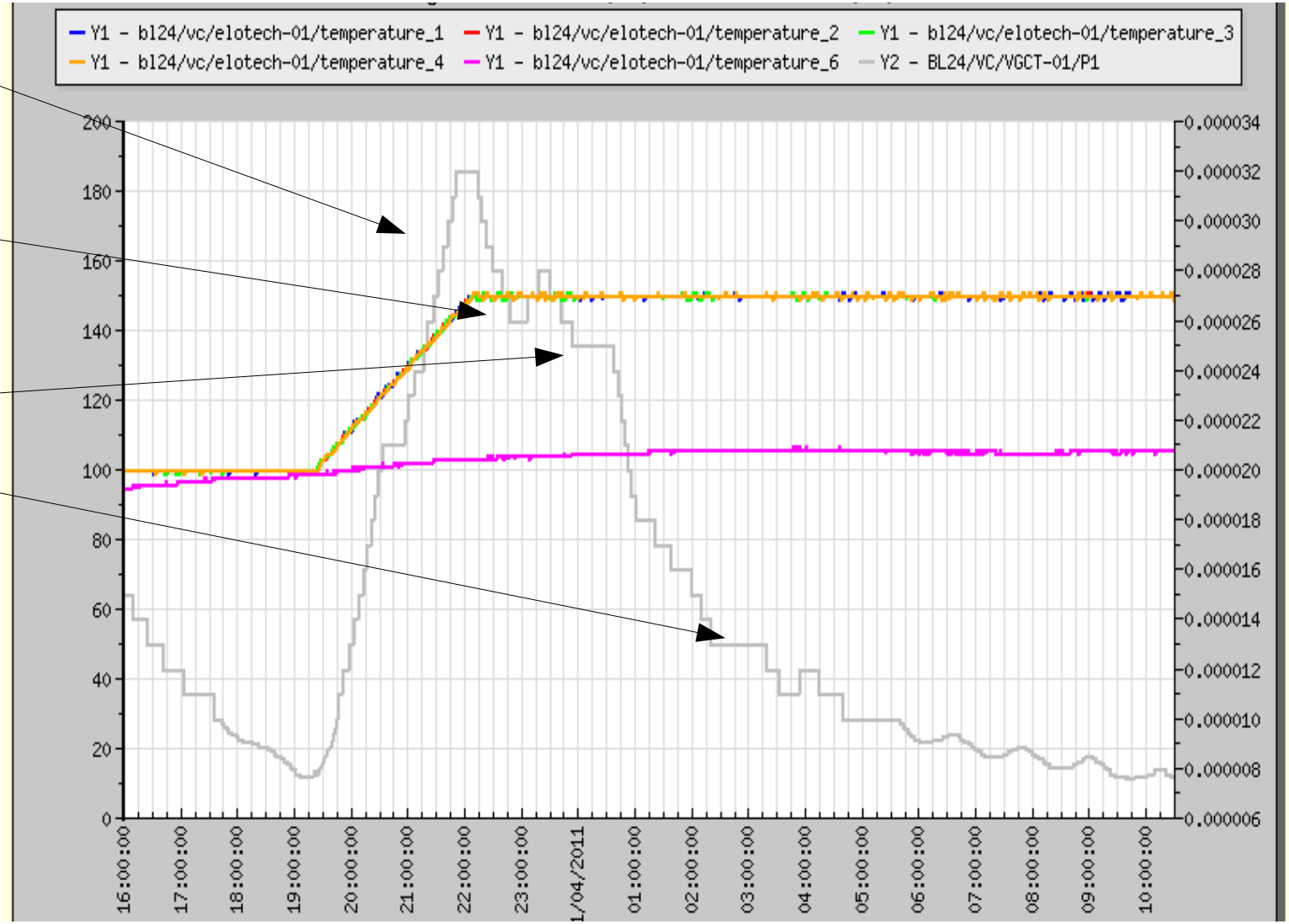
- Incidence
- Notification
- Archiving

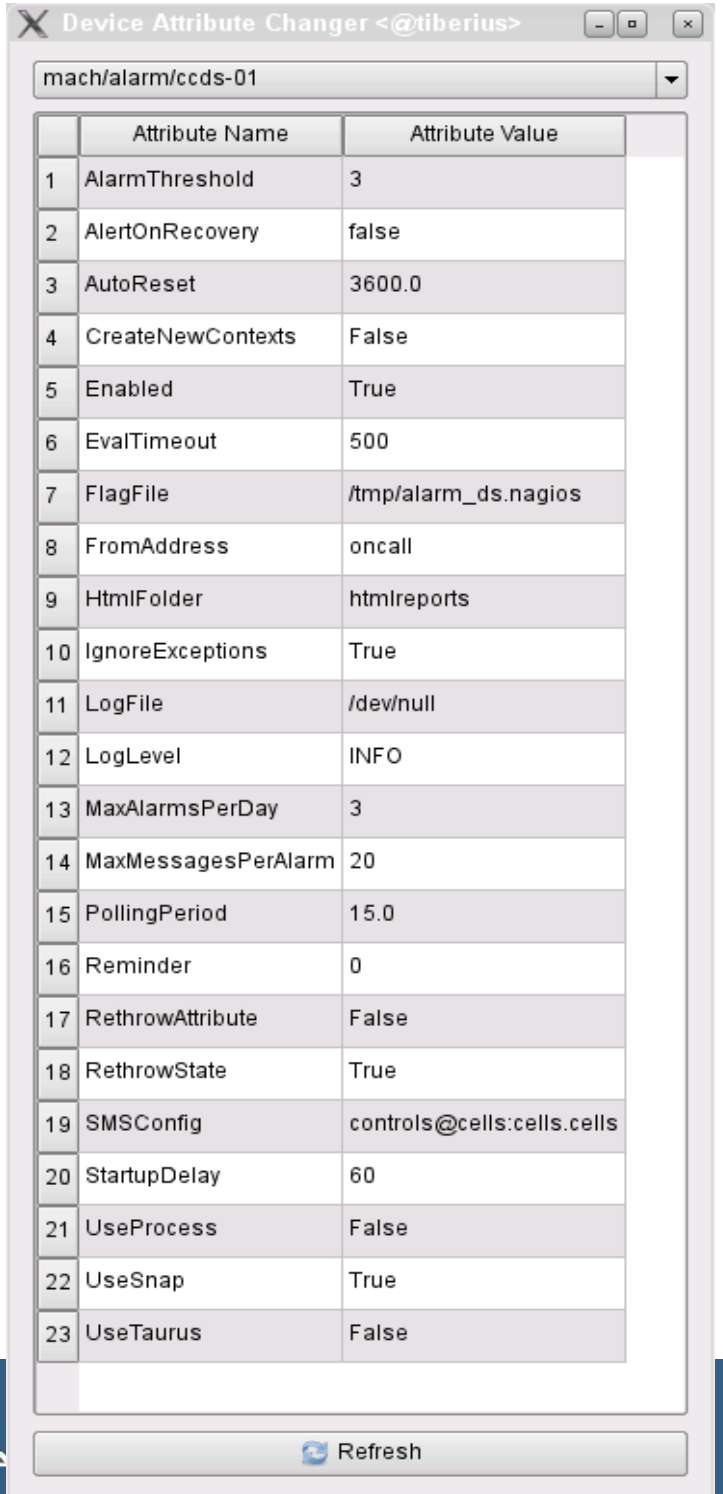
- Recovery
- Reminder

- Acknowledge

- AutoReset

- Actions?





	Attribute Name	Attribute Value
1	AlarmThreshold	3
2	AlertOnRecovery	false
3	AutoReset	3600.0
4	CreateNewContexts	False
5	Enabled	True
6	EvalTimeout	500
7	FlagFile	/tmp/alarm_ds.nagios
8	FromAddress	oncall
9	HtmlFolder	htmlreports
10	IgnoreExceptions	True
11	LogFile	/dev/null
12	LogLevel	INFO
13	MaxAlarmsPerDay	3
14	MaxMessagesPerAlarm	20
15	PollingPeriod	15.0
16	Reminder	0
17	RethrowAttribute	False
18	RethrowState	True
19	SMSConfig	controls@cells:cells.cells
20	StartupDelay	60
21	UseProcess	False
22	UseSnap	True
23	UseTaurus	False

PyAlarm Device Properties

Enabled may be True or a time (e.g. 120 seconds) to Ignore alarms already enabled during startup.

PollingPeriod controls the frequency of update.
AlarmThreshold controls alarm triggering and .delta
AutoReset time will reset the alarm if condition recovers

Reminder/AlertOnRecover for extra notifications
FlagFile/LogFile/HtmlFolder controls local logging.

Alarm history controlled by CreateNewContexts/UseSnap
UseEvents replaced by UseTaurus; which is set to False by default

IgnoreExceptions/RethrowAttribute/State control whether exceptions should trigger alarm or not or be replaced by None.

UseProcess (python subprocess) still under development

Alarm Life Cycle

- Condition: CIRCE_PRESSURE:BL24/VC/VGCT-01/P1>3e-5

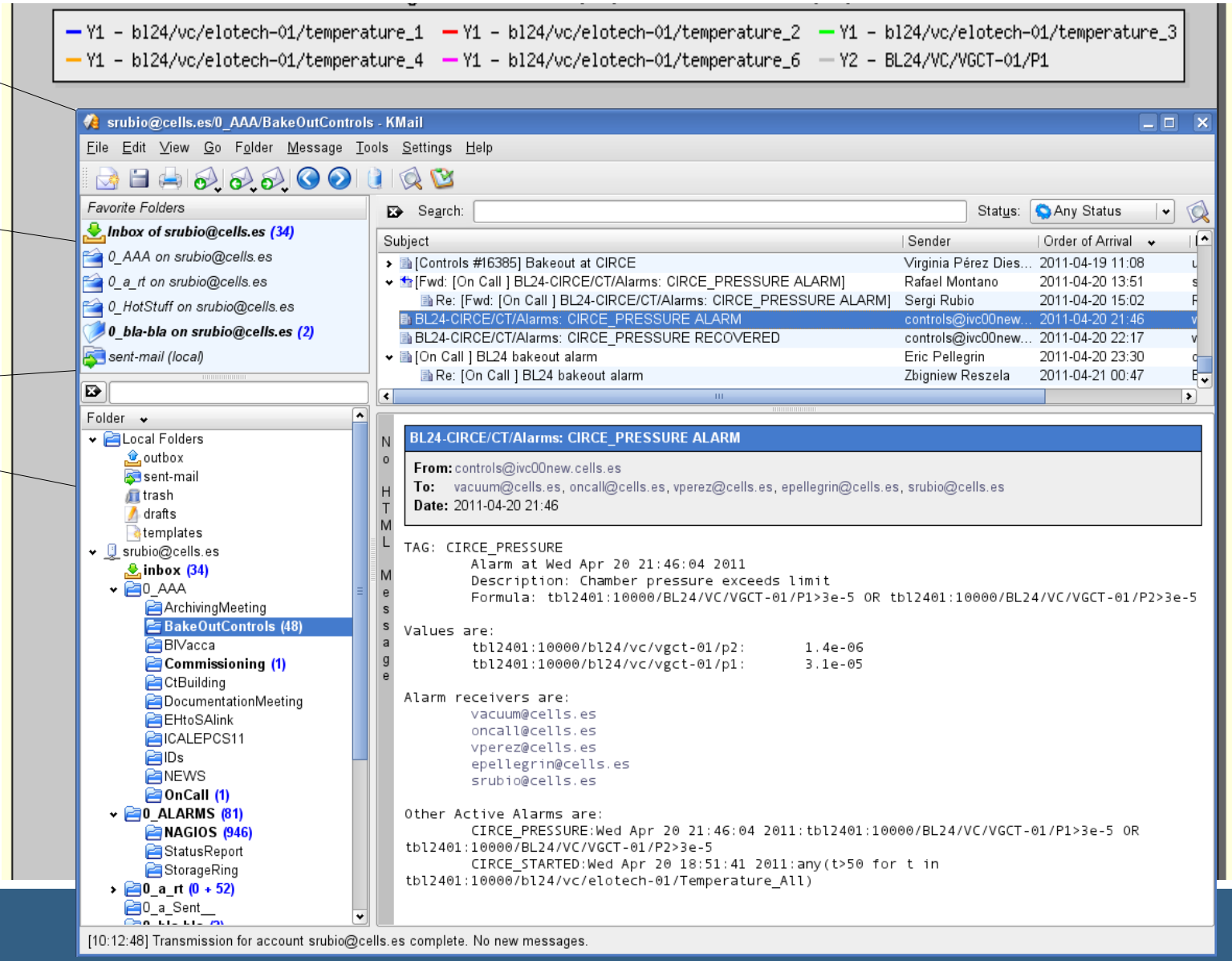
- Incidence
- Notification
- Archiving

- Recovery
- Reminder

- Acknowledge

- AutoReset

- Actions?



Declaring Receivers in Phonebook

Receivers can be set in AlarmReceivers property. When using tags previously declared in PyAlarm.Phonebook class, theneach tag will be replaced by its mailing list..

%BEEP:ACTION(alarm:command,mach/alarm/beep/play,$DESCRIPTION)

%CONTROLROOM:operators@cells.es,SMS:+34646291497

%CTRLMV:oncall@cells.es,SMS:+34682793983

%CTRL3:oncall@cells.es,SMS:+34682793983

%PLCMV:plc@cells.es,SMS:+34638420276

Receivers: Executing Actions and/or External Notifications

AlarmsList:

BL_FE_OPEN:
bl/ct/plc-01/FE_AUTO and
host:10000/chan/ct/fe/value and
bl/ct/plc-01/BL_READY and not bl/ct/plc-01/fe_open and
not bl/ct/-plc-01/fe_control_disabled

AlarmsReceivers:

BL_FE_OPEN:
ACTION(**alarm:attribute**,bl/ct/plc-01/OPEN_FE,1),

ACTION(**alarm:command**:test/notif/blmachine/**popup**
,**$ALARM,$DESCRIPTION**,15)

Related Projects

ProcessProfiler
- Provides CPU stats (cpuUsage, memUsage, ...)
- Can be used to trigger alarms (complementary to Nagios)

FestivalDS
- Beeping (using OS)
- **Speech synthesizer** (using festival linux package)
- Beep+Speech
- **Pop-up** notifications, using libnotify
- FestivalDS must run with the same user that is managing the desktop

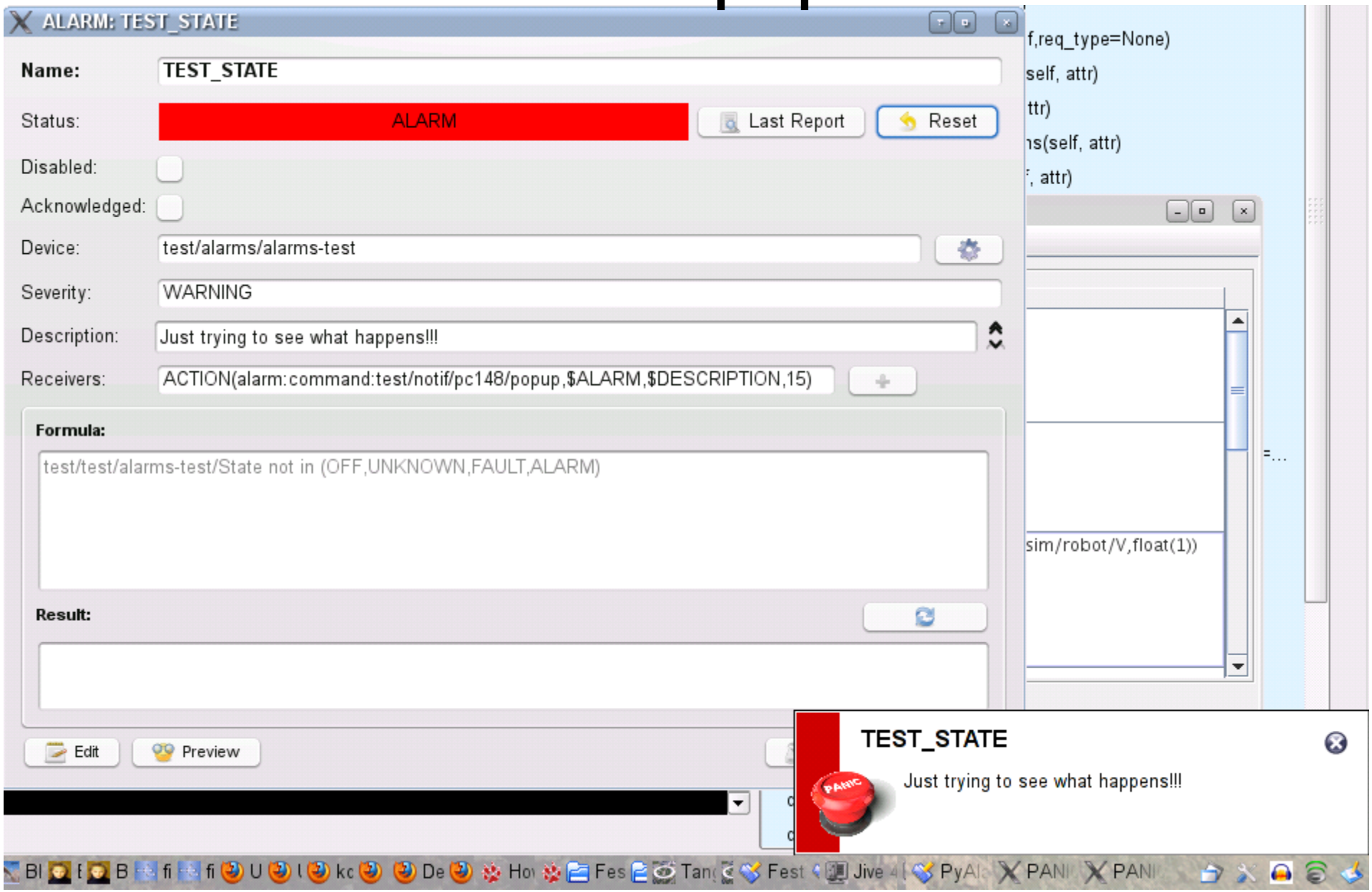
e.g. ACTION(alarm:command:test/notif/controls01/popup,$ALARM,$DESCRIPTION,15)

Fandango.tango.*

- Caseless **regular expression** parsing/sorting/matching (cached, offline when possible)
  e.g. get_matching_[device/attributes/labels/alias/properties/hosts/...](regexp)

- TangoEval (evaluation of alarm-like code; with user-macros like FIND or GROUP)
- Smart singletons: TangoCommand, CachedAttributeProxy, TangoedValues
- ProxiesDict when UseTaurus = False

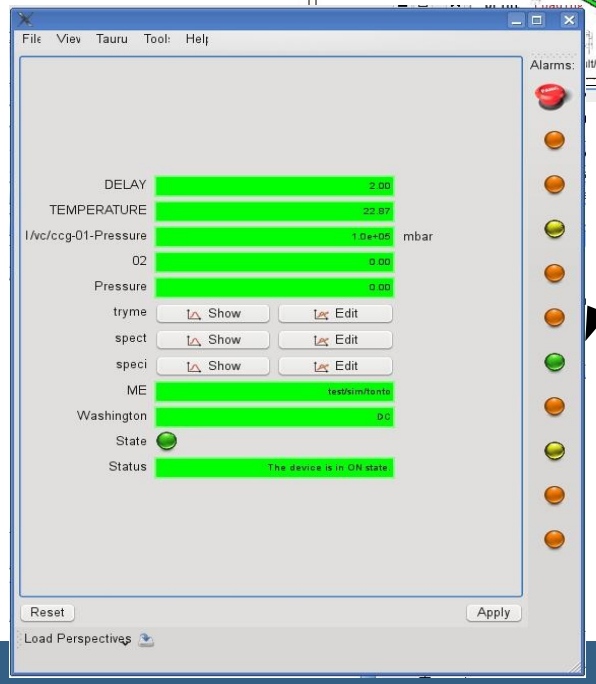
Alarm Pop-ups

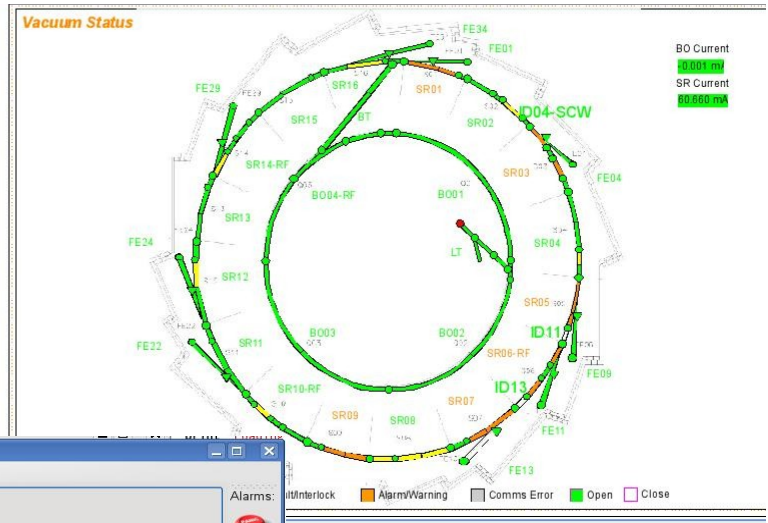


TODO ...

- Multiprocess
- Use alarms to switch synoptics
- Use own DB for configuration and logging
- Persistent notification history
- Easier alarm-on-dropped-attributes-quality
- Events receiving/filtering/pushing
- Alarms Expiration Date (now using T(date))
- PhoneBook System-Wide
- Alarm based config instead of Device
- Taurus Toolbar (best way to compact information?)
- Web-based config tool

PANIC still evolving ...





Summary

- **PyAlarm** running at ALBA **since 2007**
- Panic UI deployed in the machine **since 2011**

- Panic is both **Distributed and System-consistent**
- **DB optional** (may work as a no-db device server)
- Dynamic, formulas and **devices modifiable online**
- **Regular expression** matching
- Full Attribute value access (value, time, **quality, delta, exception**)
- Selective **exception management**

- **PyAlarm triggers** 4 different messages/actions:
- ALARM, REMINDER, ACKNOWLEDGE, RESET

- It also uses 4 different **severities**:
- INFO, WARNING, ERROR in DEBUG
- DEBUG alarms don't trigger receivers

- Logging in local files and in the **Snapshoting Database**
- **Extended** by interaction with other Tango devices.

More info and SVN links at www.tango-controls.org