PANIC, The ALBA Alarm System

Package for Alarms and Notification of Incidences from Controls

Sergi Rubio Manrique, ALBA Synchrotron, Barcelona

Some Alba Facts

- Less than 15 km. from Barcelona downtown.

- 3 Gev Lightsource, in operation since 2010
- 7 Beamlines open to users since 2012, 2 more in construction.

- Member of the Tango Collaboration.
- 4000+ Tango Devices, 160+ IOC's, 40+ virtual machines
- 90% of our devices are Python devices (PyTango)
- 99% of our graphic applications are PyQt (TAURUS)

- In active collaboration with all members of the Tango community in many projects:
  Tango, PyTango, Taurus, Sardana. Tango Archiving, Lima, Icepap, MxCube, **PANIC**

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.

Panic Alarm Life Cycle

- TAG/Description: CIRCE_PRESSURE, Beamline pressure is high
- Receivers: circe@cells.es, SMS:+34333222111, ACTION(...
- Condition: CIRCE_PRESSURE:BL24/VC/VGCT-01/P1>3e-5

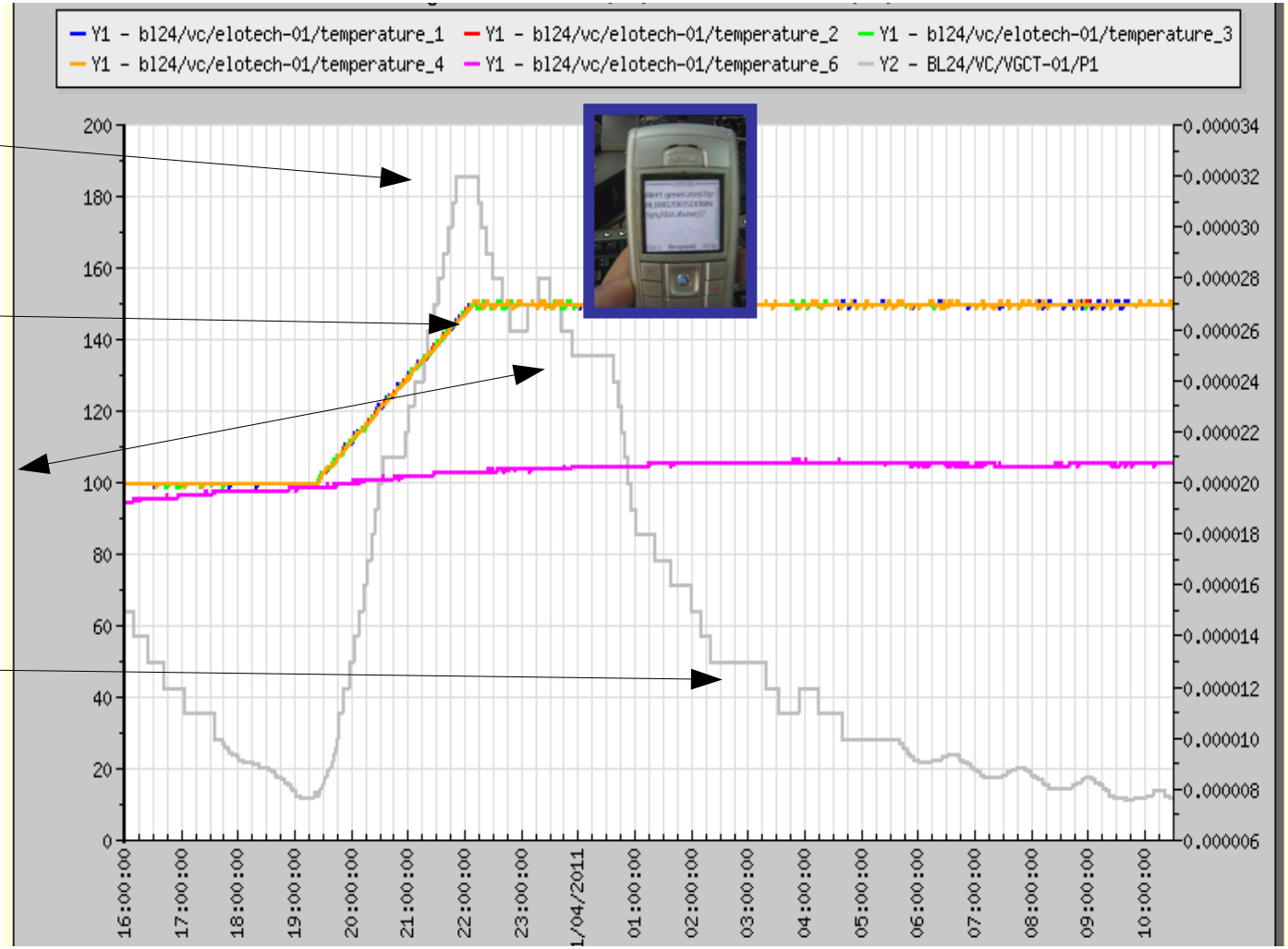
- Incidence
- ALARM
- Logging

- RECOVER
- REMINDER

- ACKNOWLEDGE

- RESET (Auto)
- Logging

- Further Actions?

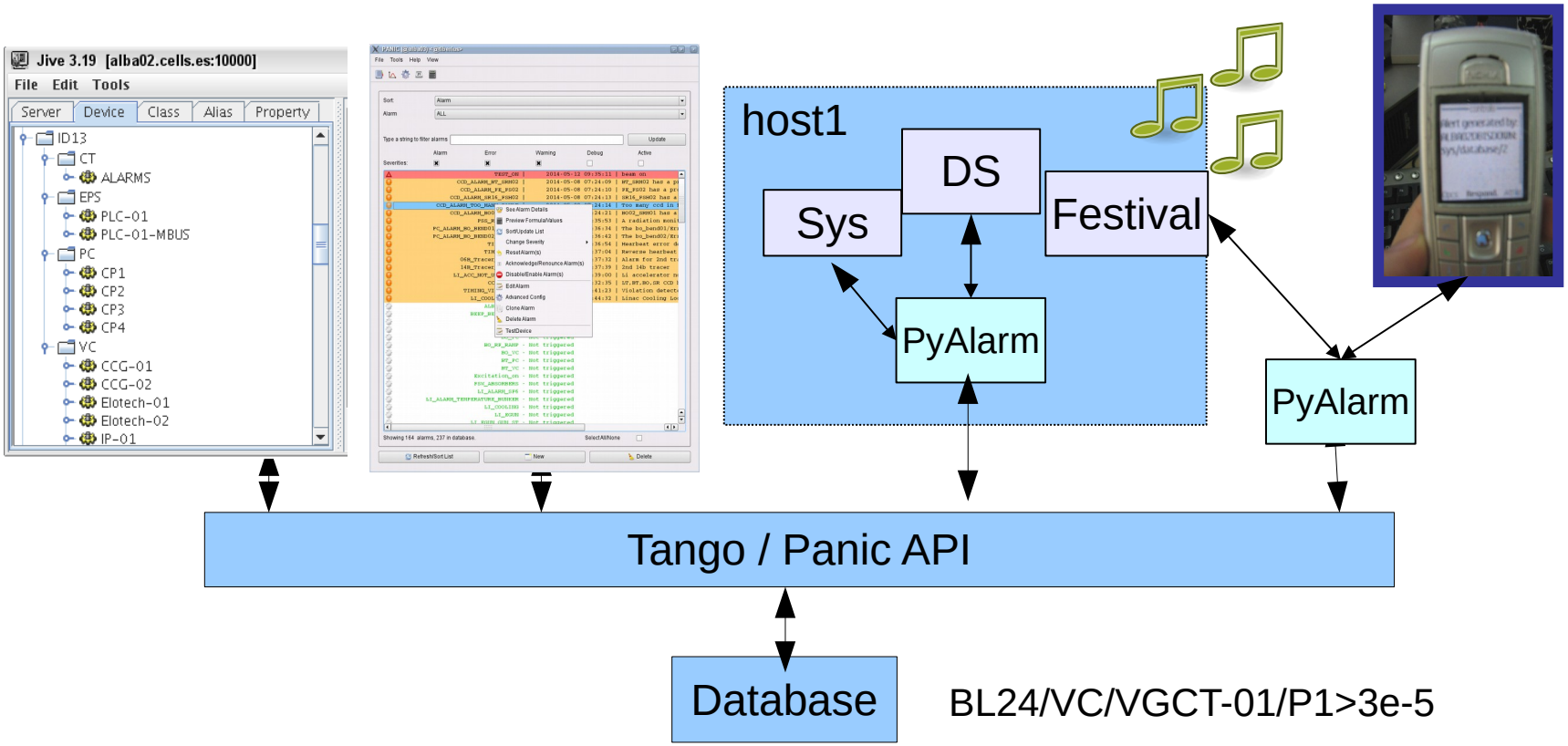


Package for Alarms and Notification of Incidences from Controls

- Based on **PyAlarm** Tango Device
- Each device evaluates a list of pseudo-**python** formulas
- Triggers actions/notifications associated to each alarm
- Evaluation is tuneable for each PyAlarm device
- Boolean attributes created for each Alarm, available to Archiving and other clients.
- Panic Client allow to manage,log and test alarms (2011)
- Mostly used by Vacuum and Accelerators groups.
- In operation at ALBA since 2007, 1200+ alarms declared.
- Running at MaxIV Linac since 2014.

PANIC, The Alba Alarm System

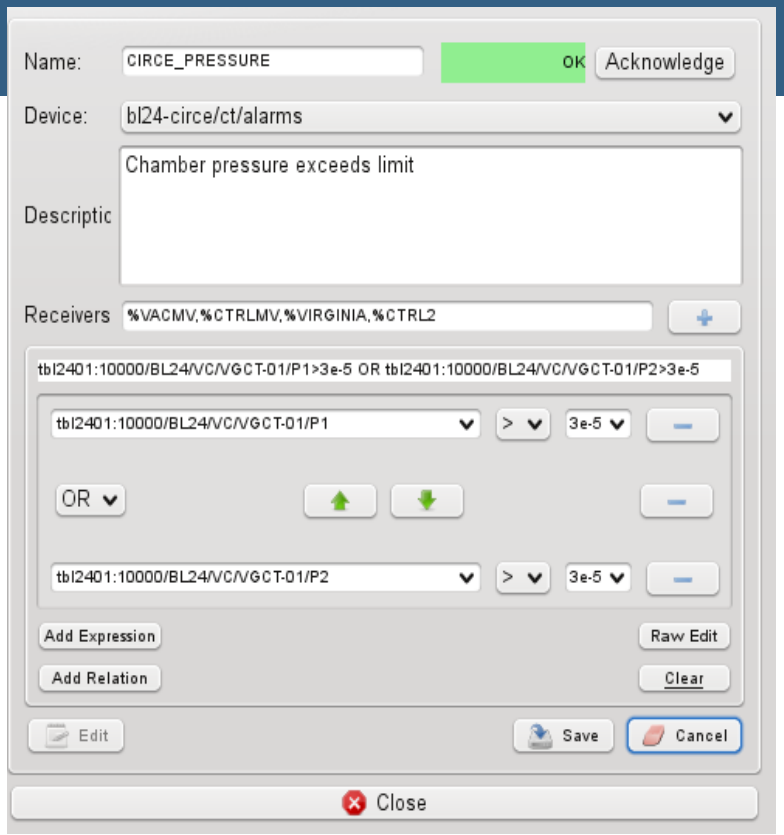
- Distributed in PyAlarm Device Servers, Panic **API provides a single view** of the system.
- Devices within a PyAlarm Server share the API and Eval objects, including caches.
- Configuration is stored in the **Tango Database,**common for all Alarms of a same device.
- Each PyAlarm device performs locally **both Logging and Notification (email/SMS)**
- **Additional actions** are passed **to external devices** (SnapArchiver, Speech, Pop-ups)
- Persistent **Alarm logging** is stored in the Tango **Snapshoting database**.



Declaring Alarms

Each Alarm is defined by:
- TAG,
- Formula,
- Description,
- Receivers,
- Severity,
- DeviceConfiguration

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



Alarm formulas:

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

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

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

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

CIRCE_CPU: max(sys/profile/tbl2401/load_avg)>0.5

CIRCE_ERROR: BL24/VC/VGCT-01/State.exception **OR**
BL24/VC/VGCT-01/State.time < now-3600

Declaring Alarms with Regular Expressions

Regular expressions allow to apply wildcards to “Extended” Tango Attribute URLs:

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

FIND,GROUP are pre-parsing Macros, applying in-place replacement before the formula is evaluated.

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_*)

But!, Protect yourself against FIND(*) queries!

Alarm Receivers and Notifications

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

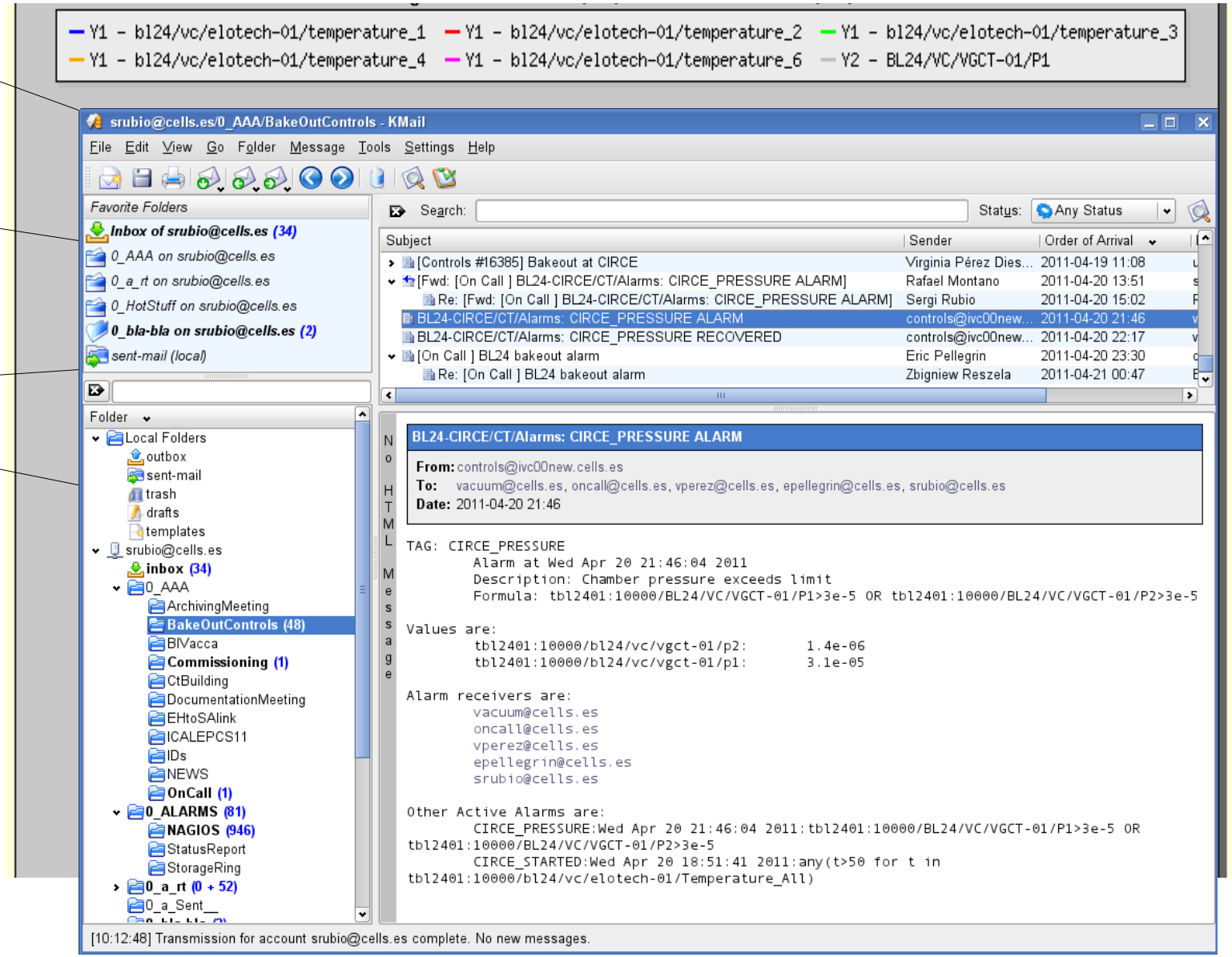
- Incidence
- Notification
- Archiving

- Recovery
- Reminder

- Acknowledge

- AutoReset

- Actions?



Declaring Receivers in Phonebook

Receivers are stored in the AlarmReceivers Device property.

Aliases for most common used receivers can be created using the PyAlarm.Phonebook Class Property in the Tango Database.

Default formats are: email, SMS, ACTION(command/attribute, ...)

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

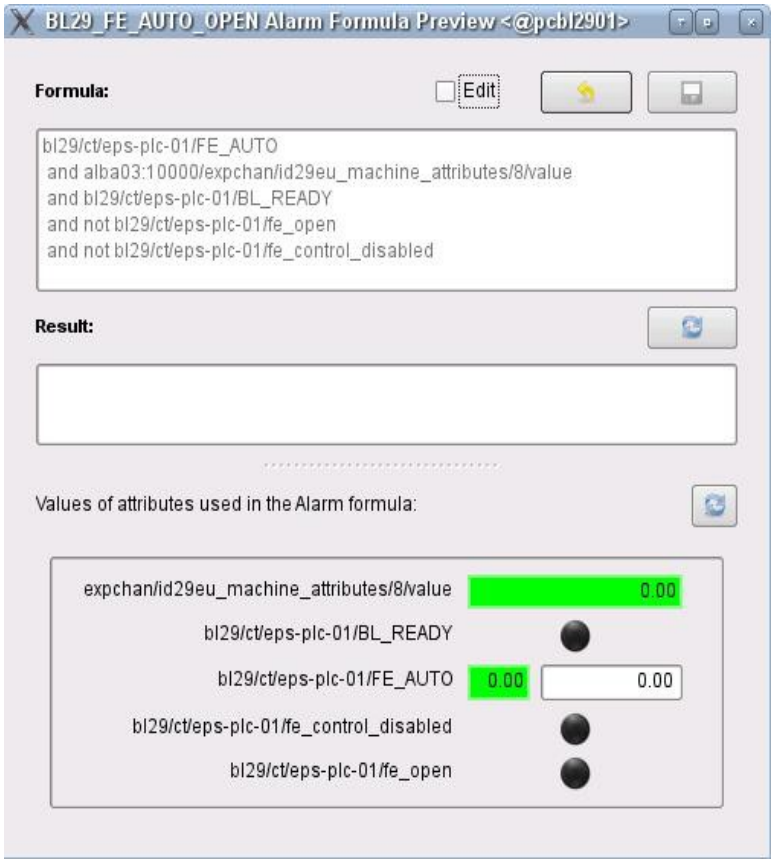
%CONTROLROOM:operators@cells.es,SMS:+34646.....

%CTRLMV:oncall@cells.es,SMS:+34682.....

%CTRLBL:oncall@cells.es,SMS:+34682.....

%PLCMV:plc@cells.es,SMS:+34638....

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

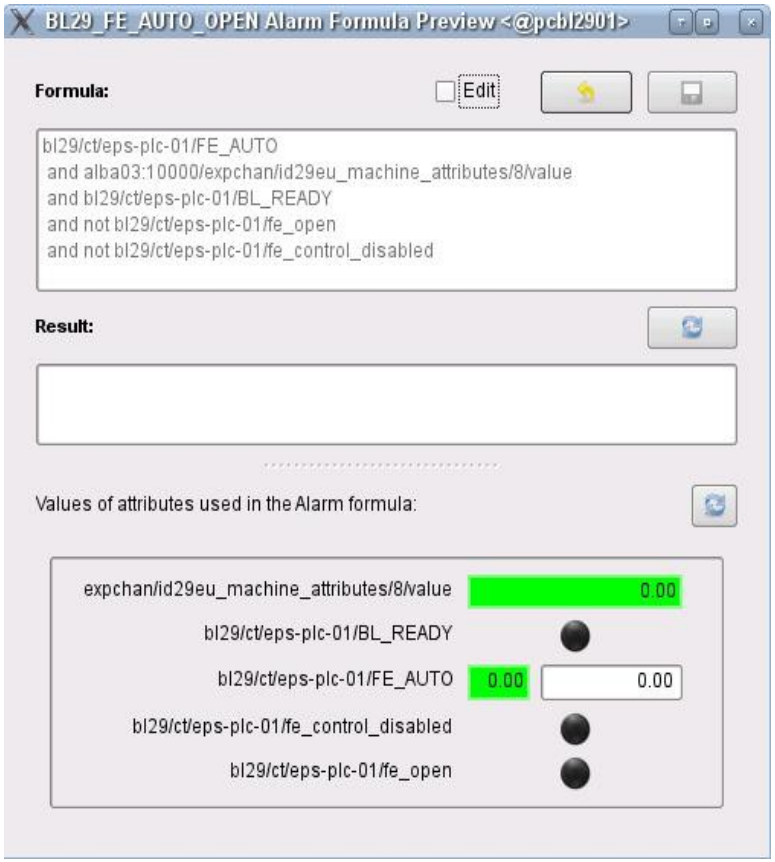
AlarmsReceivers:

BL_FE_OPEN:

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

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

Executing Actions and/or External Notifications



AlarmsList:

BL_FE_OPEN:

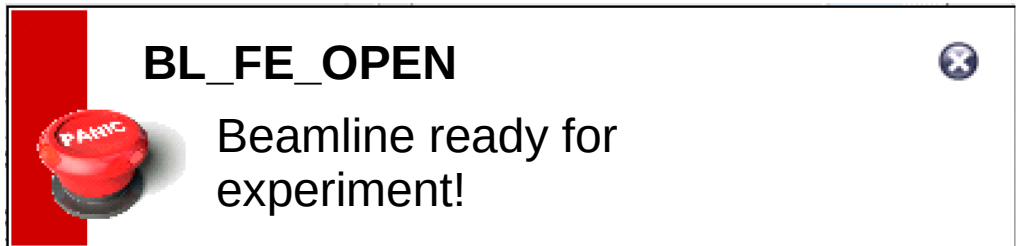
bl/ct/plc-01/FE_AUTO and
host:10000/chan/ct/fe/value and
bl/ct/plc-01/BL_READY and
not bl/ct/plc-01/fe_open and
not bl/ct/-plc-01/fe_control_disabled

AlarmsReceivers:

BL_FE_OPEN:

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

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



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.

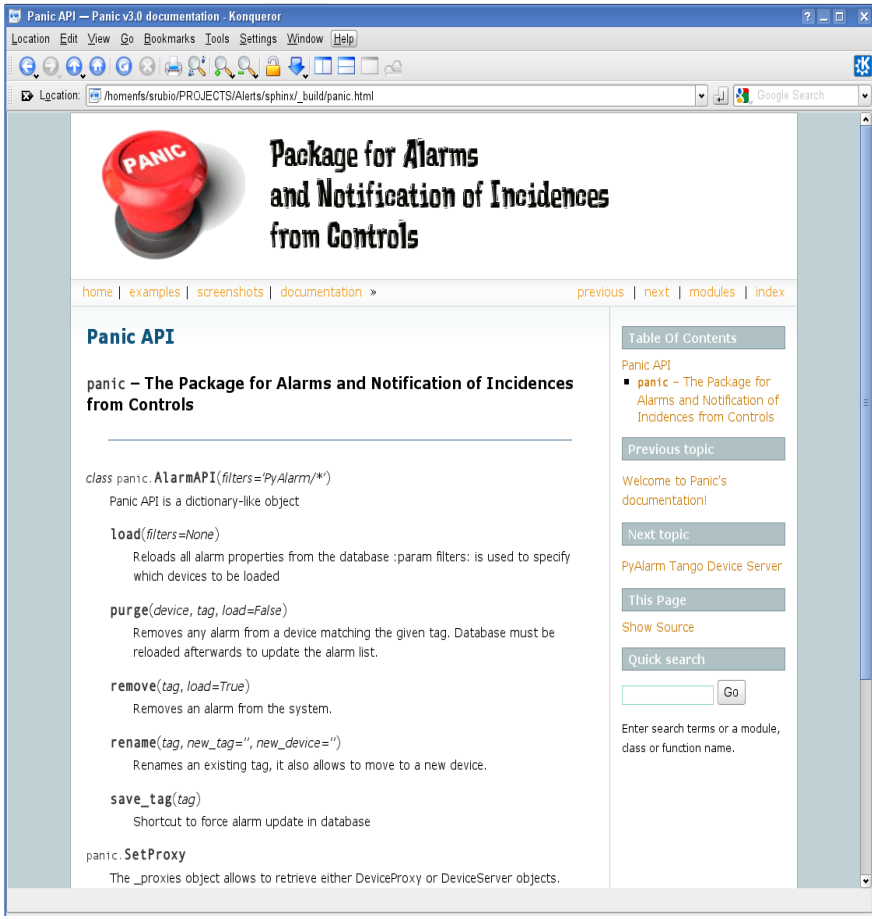
Thanks to Panic API the device server and the GUI are completely independent from the database. It will allow us to adapt our servers to some tools developed by Elettra, Soleil, Max IV, ...

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

PANIC API

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
 …]
```

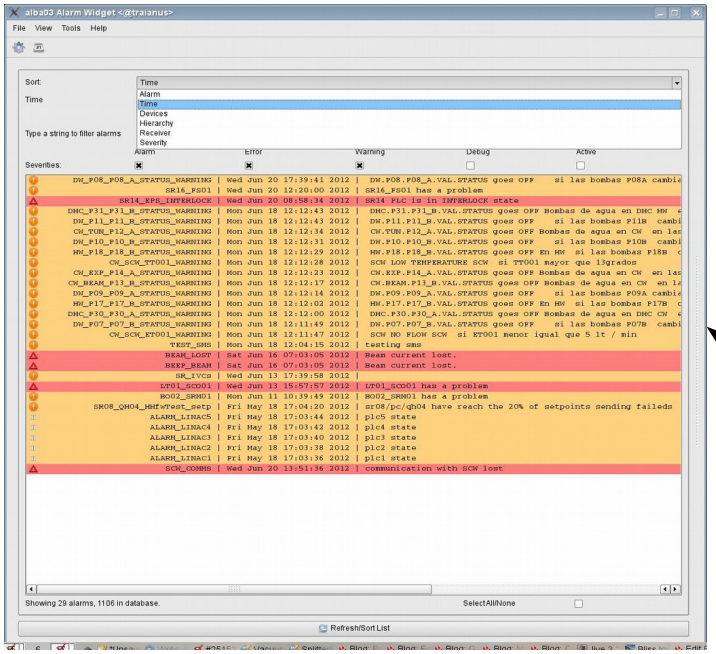
```
alarms.get(receiver='eshraq')
Out[234]:
[Alarm(RF_LOST_EUROTHERM:),
 Alarm(OVEN_COMMS_FAILED:Oven temperatures not updated in the last
5 minutes),
 Alarm(RF_PRESSURE:The pressure in the cavity exceeds Range),
 Alarm(OVEN_TEMPERATURE:The Temperature of the Oven exceeds
Range),
 Alarm(RF_EUROTHERM:),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

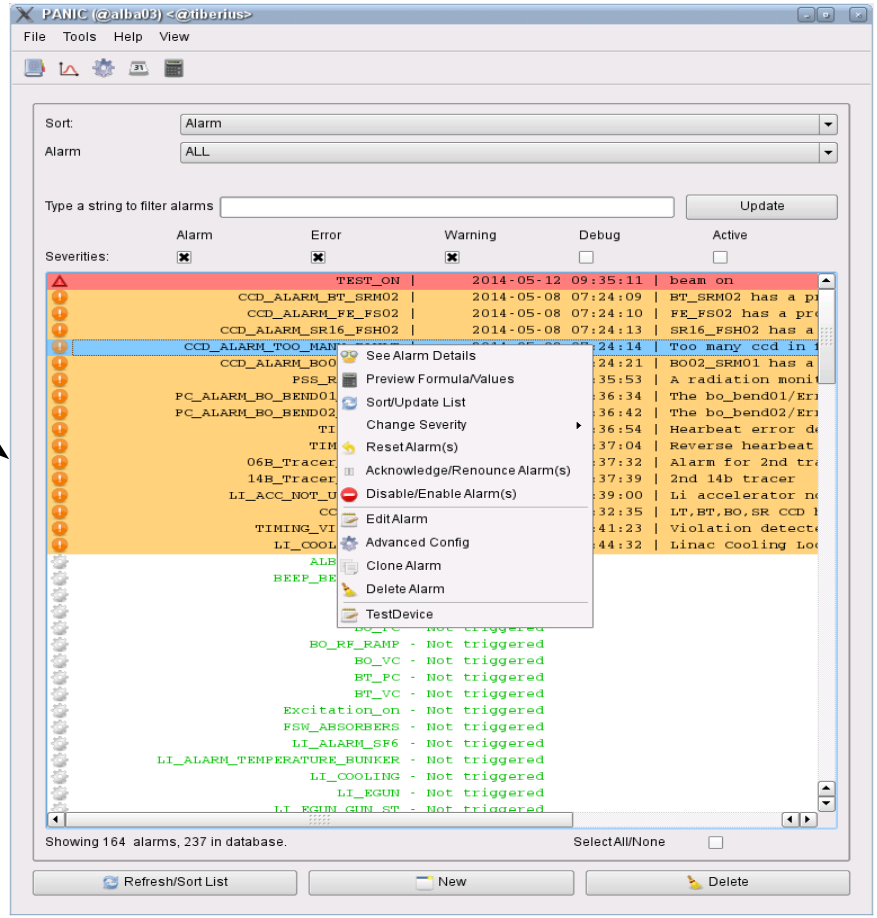
Panic User Interface

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.

Alarms are sorted by severity: ERROR, ALARM, WARNING, INFO, DEBUG

A text search is also provided that allow to locate alarms by any of the attributes used in formula or words used in description (soon available as Taurus Search Bar).

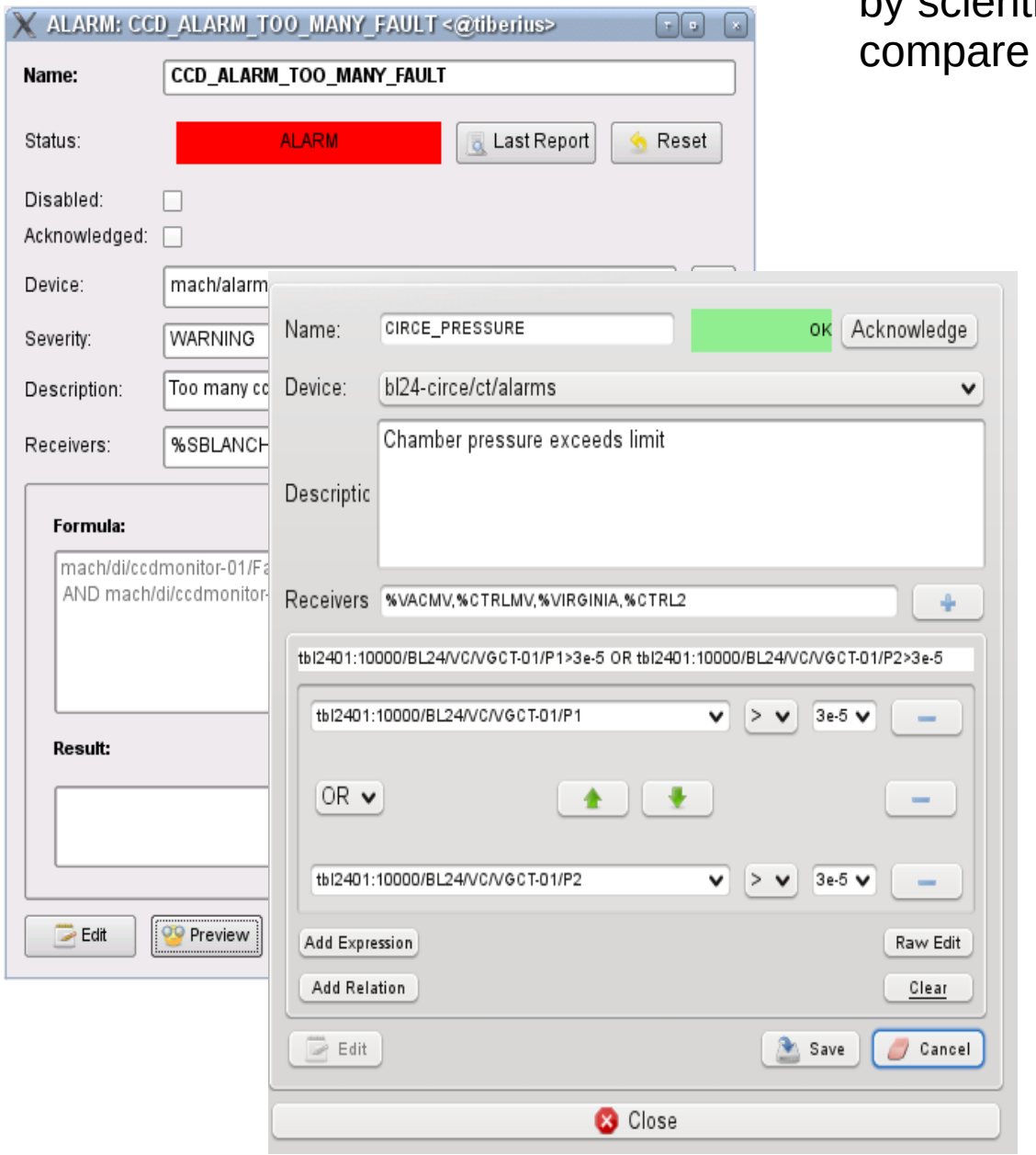


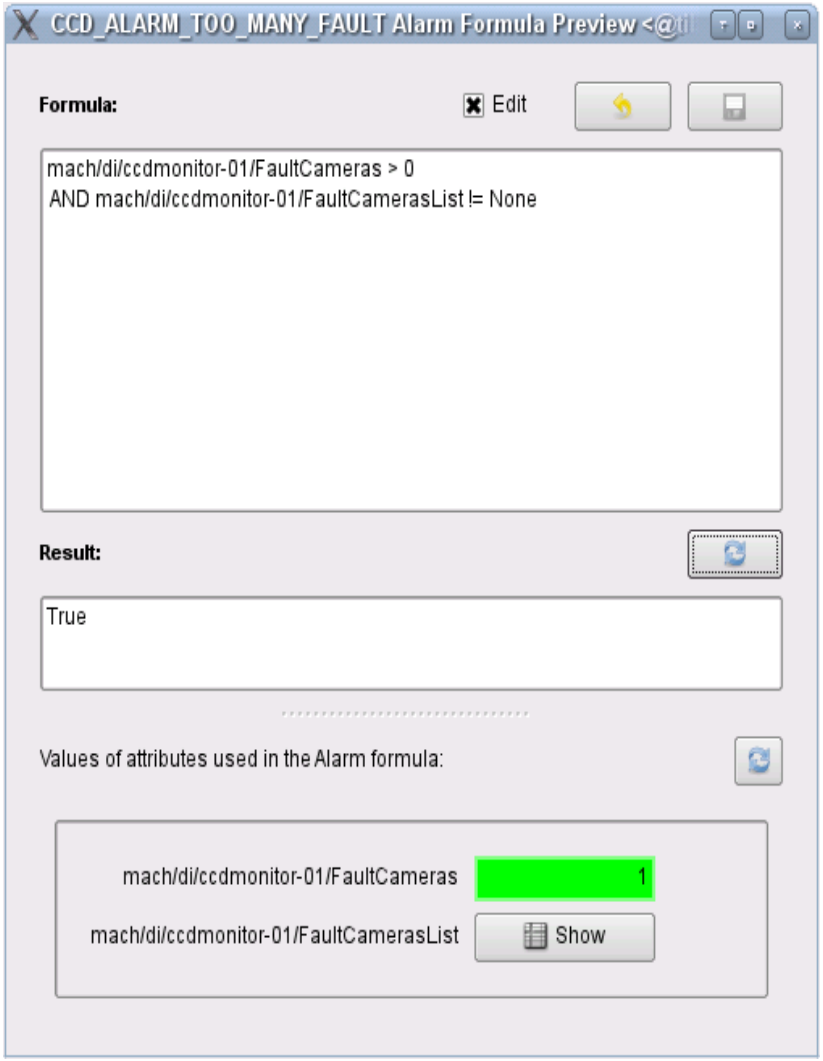


For each alarm the menu allows to Configure, Reset the alarm or show the attribute values that triggered it.

UI Editor and Preview

The preview panel is an independent widget, used by scientists as a Tango calculator that allows to compare formula against current system values.



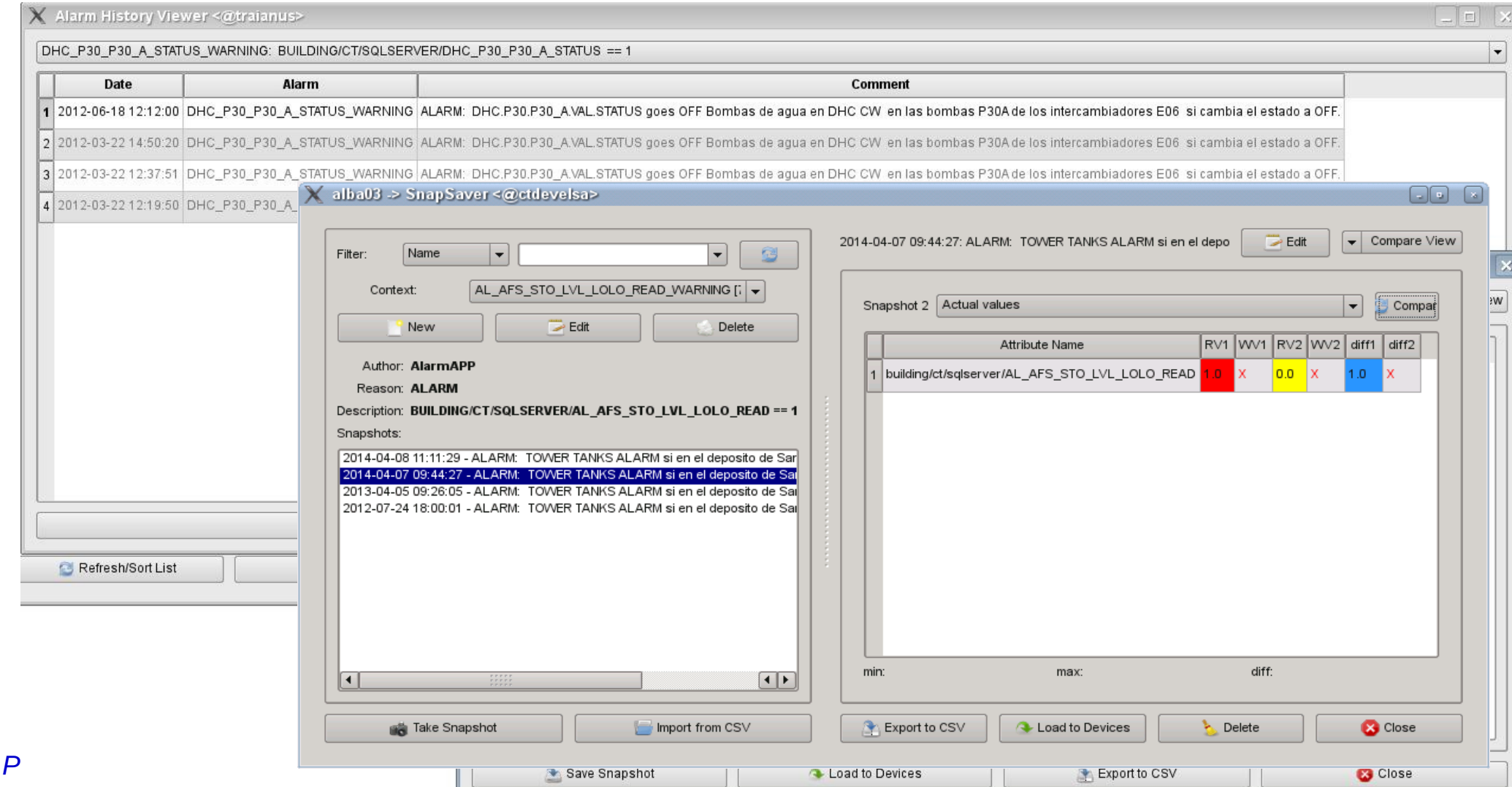


Record/View Alarm History using Snaps

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.



PyAlarm Tuning: Device Properties

Device Attribute Changer <@tiberius>

mach/alarm/ccds-01

	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

Refresh

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

Enabled can be set to True/False or a time(e.g. 120) to ignore alarms already enabled during startup.

Reminder/AlertOnRecover for extra notifications

Alarm history controlled by CreateNewContexts/UseSnap
UseTaurus allows to delegate polling/event management to Taurus.core library. (http://sourceforge.net/p/sardana)

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

UseProcess (evaluate formulas in background processes) \ still under development

FlagFile, HtmlFolder, LogFile : alternative logging that can be recorded in the local filesystem or NFS mounted folders.

Panic Alarm Life Cycle

- TAG/Description: CIRCE_PRESSURE, Beamline pressure is high
- Receivers: circe@cells.es, SMS:+34333222111, ACTION(...
- Condition: CIRCE_PRESSURE:BL24/VC/VGCT-01/P1>3e-5

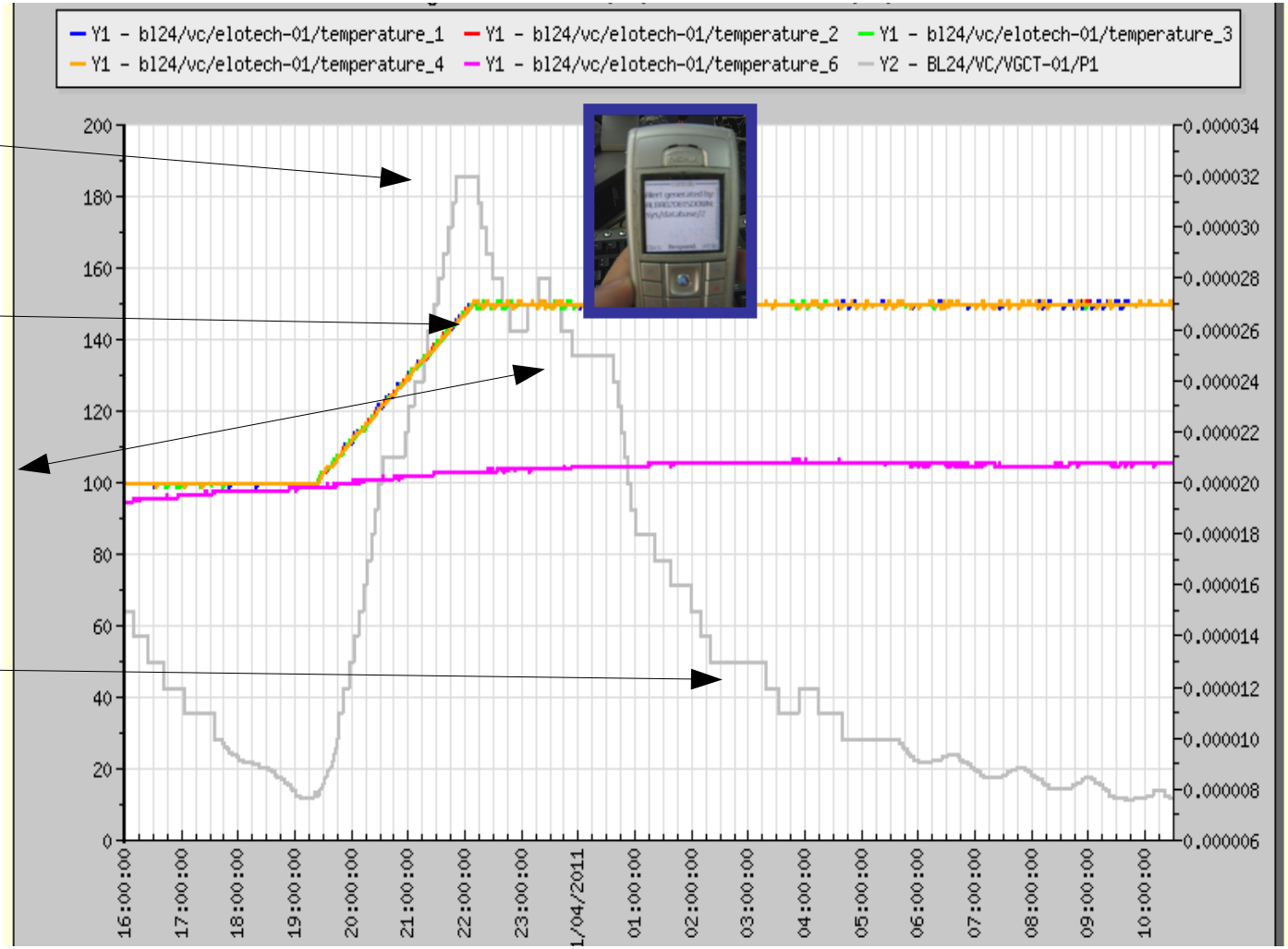
- Incidence
- ALARM
- Logging

- RECOVER
- REMINDER

- ACKNOWLEDGE

- RESET (Auto)
- Logging

- Further Actions?



Related Projects

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

FestivalDS
- Beeping (using OS services)
- **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

PyAlarm, PANIC, UI, and these devices are available at:
http://sourceforge.net/p/tango-ds/code/

ACKNOWLEDGE

- **S**oleil and Elettra institutes for their Alarm and Archiving Systems, that inspired PANIC.
- The ALBA Accelerators division for the involvement in the development of the PANIC UI
- ALBA Vacuum section for their intensive usage of early PyAlarm
- MaxIV for their development of plugins for PyAlarm and Fandango

Pending TODO's in Panic:

- Multiprocess
- Use alarms to navigate clients.
- 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 Alarm Toolbar
- Web-based config tool

PANIC still evolving ...