
PANIC training materials

Release 0.9

Piotr Goryl (S2Innovation), Tango Community

Feb 22, 2021

CONTENTS:

1	PANIC ALARM SYSTEM	1
1.1	References	1
1.2	Architecture	1
1.3	Alarm states	4
2	PyAlarm Properties Quick Reference	7
2.1	Global Properties	7
2.2	Class Properties	7
2.3	Device Properties	8
3	Annunciators/Recivers	11
3.1	Actions	12
3.2	PhoneBook	12
3.3	Global Recivers	13
4	Notes on alarms formulas	15
4.1	Tango names resolving	15
4.2	Special Keys/Macros	15
5	Integration with external systems	23
5.1	Mail	23
5.2	SMS	24
5.3	A Text Talker	25
5.4	eLog or other web-based systems	25
5.5	Knowledge database	26
6	MAX-IV ELK stack application	29
6.1	A demo of MAX-IV deployment	29
6.2	MAX-IV app description	30
6.3	Deployment	30
7	PANIC source-code	33
7.1	alarmapi.py	33
7.2	properties.py	33
7.3	PyAlarm.py	34
8	Deployment and configuration	35
8.1	PyAlarm dependencies	35
8.2	PANIC GUI dependencies	35
8.3	Number of devices/device servers	36
8.4	Timing configuration	36

8.5	Exceptions handling configuration	38
8.6	UseTaurus	38
8.7	Naming conventions	38
9	PANIC GUI	39
9.1	PANIC GUI description and goals	40
9.2	GUI overview	40
9.3	Alarm edit/details panel	46
9.4	Configuration of the alarms list	58
9.5	Context menu	65
9.6	Alarms management	73
9.7	Top bar menu	73
10	Alarm philosophy	79
10.1	Alarm philosophy	79
10.2	Alarm system requirements specification (ASRS)	80
11	Indices and tables	81

PANIC ALARM SYSTEM

The PANIC is software for building an alarm management system. It is developed mainly by ALBA synchrotron, however, several laboratories proposed some extensions or alternative solutions.

1.1 References

At this moment source code is kept on the GitHub: <https://github.com/tango-controls/PANIC>, however, it will be migrated to the GitLab tango-controls organization, soon.

Documentation: <https://tango-controls.readthedocs.io/en/latest/tools-and-extensions/alarm/panic.html?highlight=panic>

Papers: There are several publications referencing PANIC available on JACoW

Forum: There is a dedicated forum thread on Tango Controls: <https://www.tango-controls.org/community/forum/c/general/development/panic-the-alba-alarm-system/?page=1>

1.2 Architecture

Below is a short introduction to PyAlarm system architecture. Detail information is available in the online documentation.

A typical deployment of the PANIC system contains (in addition to Tango Controls):

- PyAlarm device servers
- PANIC GUI
- SNAP archiving

There can be multiple PyAlarm devices provided by multiple *PyAlarm* servers. A *PyAlarm* device keeps alarms configuration in its properties. Each *PyAlarm* device serves a set of alarms (alarms are assigned to a *PyAlarm* device).

A *PyAlarm* device is subscribing or polling tango attributes which appear in its alarms formulas (part of alarm configuration). Alarm formulas are periodically evaluated. If the evaluation result is *True* alarm is triggered (the *formula* is detecting an abnormal condition, according to IEC-62682). An abnormal condition is defined when the alarm formula evaluation results in *True*.

PyAlarm devices play the *Alarm Log* role in the *Alarm System*.

If configured so, a *PyAlarm* stores alarm state changes as entries in a SNAP database. The entry also contains values of tango attributes involved in the formula evaluation.

SNAP database is the *Alarm historian* role in the *Alarm system*.

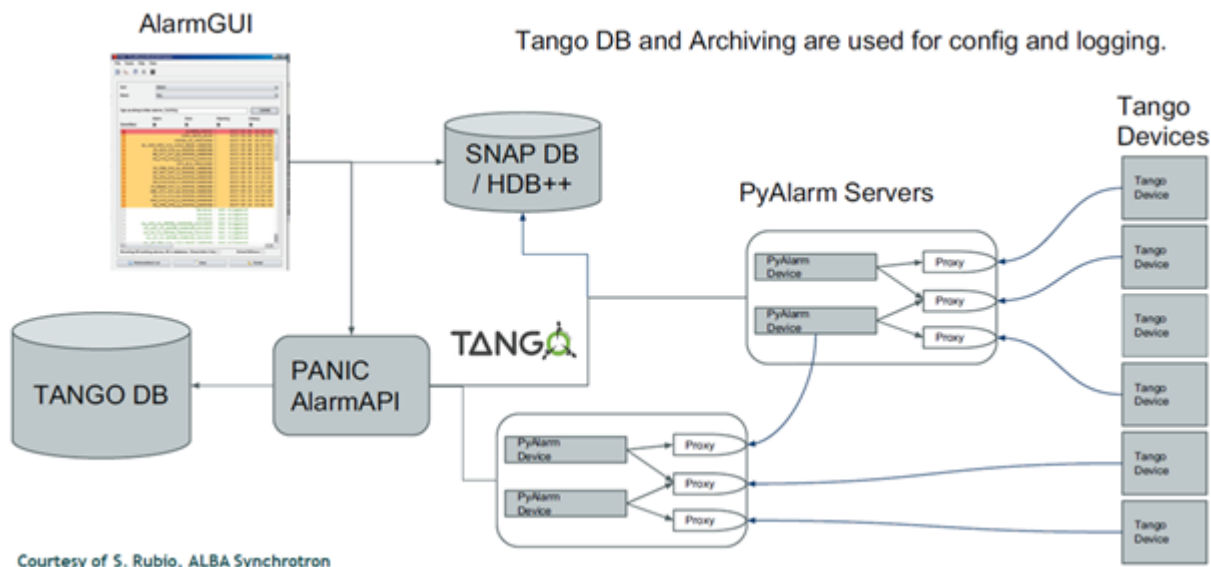


Fig. 1.1: PANIC system architecture

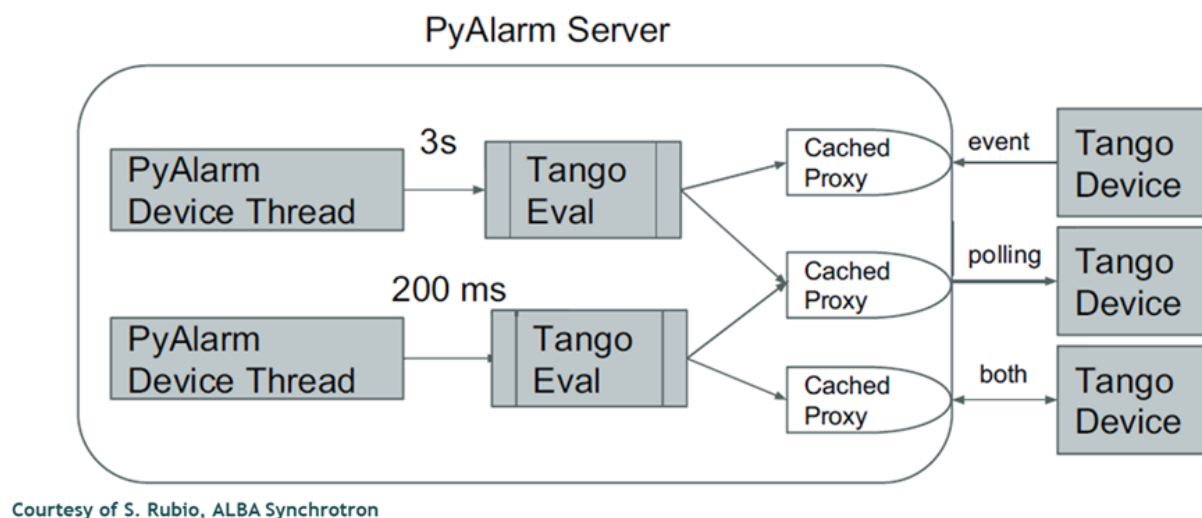


Fig. 1.2: PyAlarm internals

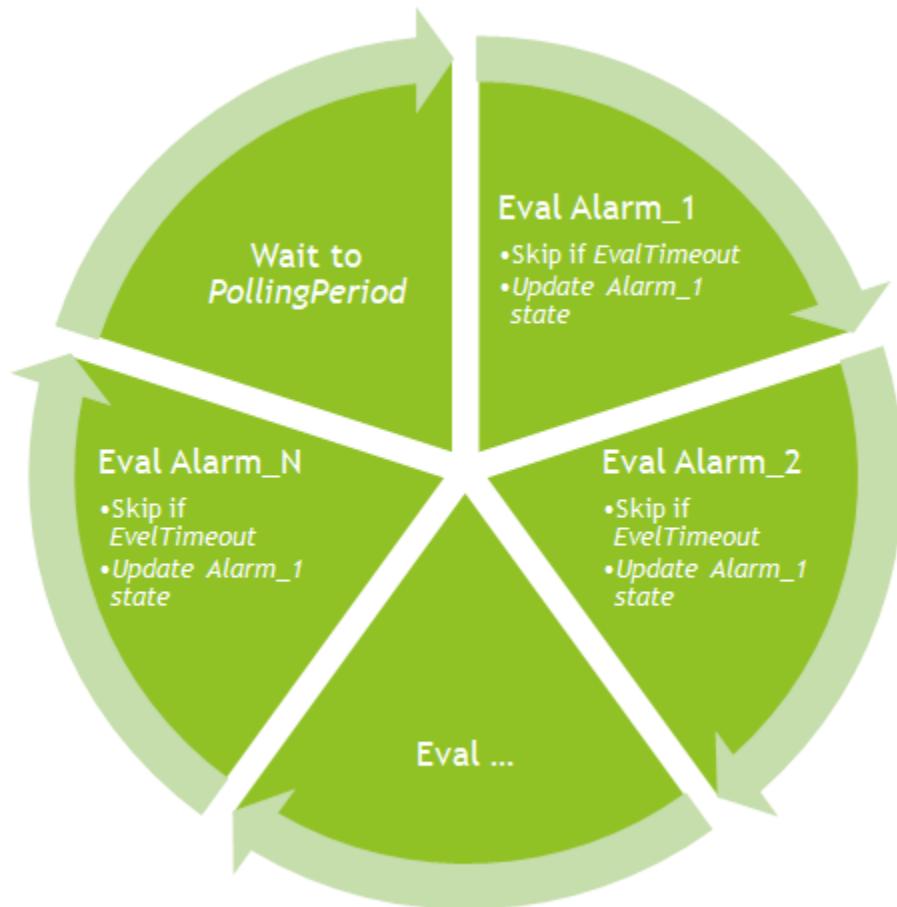


Fig. 1.3: Alarms evaluation cycle

PANIC GUI is gathering and managing information from all *PyAlarm* devices. It provides the *HMI* role according to the IEC norm.

There could be additional components in the system, including Annunciators (mail system, SMS system, ...) which are *Alarm system's External systems*.

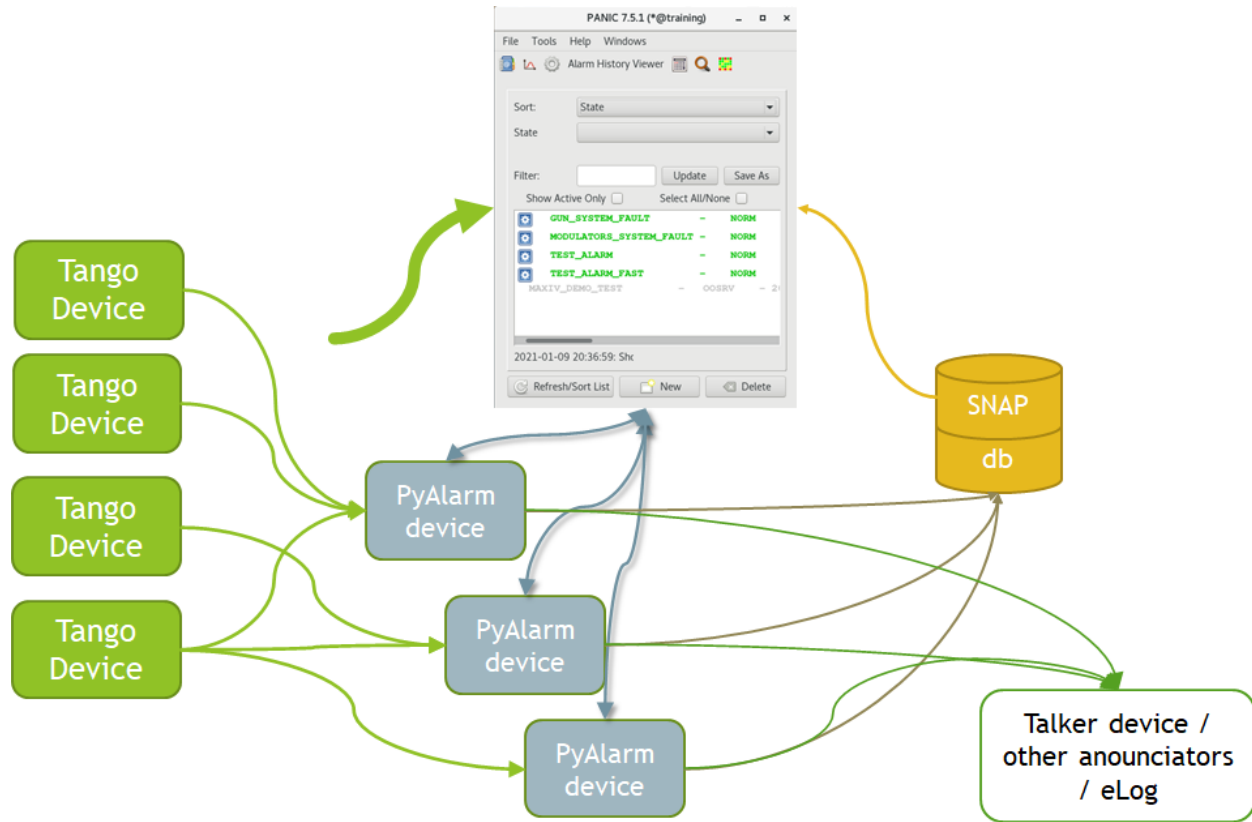


Fig. 1.4: Information flow in the PANIC system

1.3 Alarm states

The PANIC follows IEC62682 standard for alarm states.

Each alarm individually can be in one of the following states:

- Normal (NORM) - the condition for the alarm is not active (formula evaluation has returned *False*),
- Uncacknolwdged (UNACKED) - the alarm condition has become active (there is an alarm situation than needs an operator reaction),
- Acknowledged (ACKED) - the alarm (the condition) is still active but an operator has acknowledged it (react, start solving the alarm),
- Return to Normal (RTNUN) - alarm condition recovered (is not active) but the alarm has not been automatically reset or acked by any operator,
- Shelved (SHLVD) - temporary disabled by an operator to focus on more important alarms,
- Suppressed by design (DSUPR) - notification disabled by the system, for example during startup to avoid alarms flooding in startup transient states,

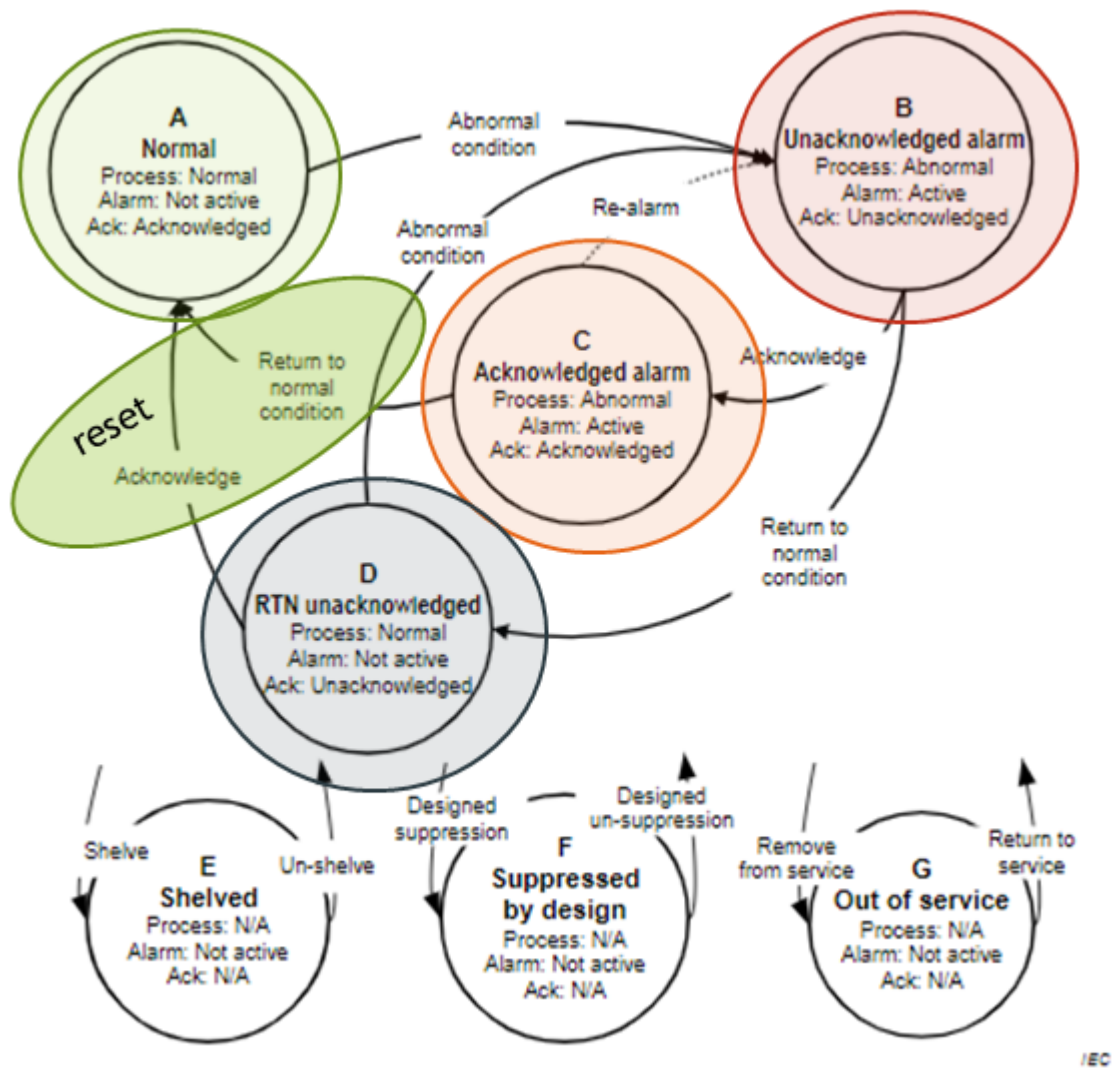


Fig. 1.5: Alarms states

- Out of service (OOSRV) - the alarm is defined in the system, but cannot be evaluated at this moment (i.e. a component which evaluates the alarm is switched off).

1.3.1 Working with alarms



Fig. 1.6: Operation with an alarm

Typical (and recommended) operation of an alarm is as follows:

- The alarm system detects an alarm condition and triggers the alarm (the alarm changes its state from NORM to UNACKED),
- An operator notices the alarm (via a GUI or via other means of annunciation),
- If the operator is going to solve the alarm situation, he acknowledges the alarm (the alarm state becomes ACKED, so another operator knows that someone is dealing with the issue),
- The operator solves the alarm cause (the alarm state changes to RTNUN),
- The operator or the system itself resets the alarm (the alarm state changes back to NORM).

PYALARM PROPERTIES QUICK REFERENCE

All properties are listed in `panic/properties.py` file: <https://github.com/tango-controls/PANIC/blob/master/panic/properties.py>

2.1 Global Properties

These properties are Tango *free properties* under PANIC group. Global Properties are used by PyAlarm and/or to store some GUI configuration.

PhoneBook: List of receiver aliases, declared like: `%USER:user@accelerator.es;SMS:+34666555666,`

2.2 Class Properties

Class properties are used applies to all *PyAlarm* devices.

SMSConfig: Arguments for `sendSMS` command,

SMSMaxLength: Maximum length of SMS messages,

SMSMaxPerDay: Maximum SMS messages per day,

MailMethod: Define mail method. Can be set to `mail` or `smtp[:host[:port]]`,

MailDashOption: If not empty, `mail` command is invoked with `-r` option to specify *from_address* (instead of `-S from=...`),

FromAddress: Address that will appear as Sender in mail and SMS,

AllowedActions: List of OS commands which alarms ``ACTION``s are allowed to execute,

StartupDelay: Number of seconds that PyAlarm will wait before starting to evaluate alarms,

PanicAdminUsers: Users authorized to modify the Alarms (apart of receivers),

PanicUserTimeout: Number of seconds to keep user login in panic GUI,

UserValidator: Module.Class to be used to validate admin user/passwords,

GlobalReceivers: Receivers to be applied globally to all alarms. Declared as:

FILTER:receiver,ACTION(MESSAGE:...), like: `*VC*:vacuum@cells.es,ACTION(RESET:command,t/t/t/stop)`,

AlarmWikiLink: An URL to a Wiki page, where one can find more info on alarms. If set it will appear on the *AlarmEditor* widget. The URL may contain a key `{%ALARM%}` which is then substituted with an alarm tag. Example: `http://wiki.cps.uj.edu.pl/alarms/{%ALARM%}`,

2.3 Device Properties

2.3.1 Alarm state/cycle related

Enabled: If False forces the device to *Disabled* state and avoids messaging; if an integer number (*N*), it will last *Disabled* for *N* seconds after Startup; if a python formula is written it will enable/disable the device according to its evaluation.

AlarmThreshold: Minimum number of consecutive formula evaluation to *True* before any alarm is triggered,

AlertOnRecovery: It can contain 'email' and/or 'sms' keywords to specify if an automatic message must be sent in case of alarm returning to a safe level.

PollingPeriod: Period in SECONDS to poll all not event-driven attributes and to run formulas evaluation. @TODO for convenience any value above 300 will be divided by 1000,

Reminder: If a number of seconds is set, a reminder mail will be sent while the alarm is still active, if 0 no Reminder will be sent,

AutoReset: If a number of seconds is set, the alarm will reset if the conditions are no longer active (RTN) for the specified time,

RethrowState: Whether exceptions in State reading will activate the Alarm,

RethrowAttribute: Whether exceptions in Attribute reading will activate the Alarm,

IgnoreExceptions: Value can be False/True/NaN to return Exception, None or NotANumber in case of read_attribute exception,

2.3.2 Historian related properties

UseSnap: If false no snapshots will be triggered (unless specifically added to annunciators with SNAP keyword),

CreateNewContexts: It enables *PyAlarm* to create new contexts for alarms if no matching context exists in the database,

2.3.3 Logging of alarms

LogFile: A file where alarms are logged, like /tmp/alarm_\$NAME.log. One can use keywords: \$DEVICE, \$ALARM, \$NAME, \$DATE. If version>6.0 a FolderDS-like device can be used for remote logging: tango://test/folder/01/\$ALARM_\$DATE.log,

HtmlFolder: File where alarm reports are saved,

FlagFile: File where a 1 or 0 value will be written depending if there is any active alarm or not. This file can be used by other notification systems,

2.3.4 PyAlarm device instance configuration

LogLevel: *stdout* log filter, like `INFO`, `DEBUG`, ...

StartupDelay: Number of seconds that *PyAlarm* will wait before starting the evaluation loop. For this time, all alarms handled by the device will have state `OOSRV` (out of service),

EvalTimeout: Timeout for `read_attribute` calls, in milliseconds,

UseProcess: To create new OS processes instead of threads (experimental),

UseTaurus: Use Taurus to connect to devices instead of plain PyTango (it is recommended to set it to `True`).

ANNUNCIATORS/RECIVERS

PyAlarm device can invoke several actions on alarm state change:

- Send an email: `user.name@domaing.us`,
- Send an SMS: `SMS:+44010101010`,
- Call a tango command: `ACTION(alarm:command, tan/go/device/command, argumets)`,
- Set a value of a tango attribut: `ACTION(acknowledge:attribute,tan/go/device/attribute, value)`,
- Call an operating system call (command): `ACTION(reset:system, '/usr/local/beep')`,

The list of anouncianttions for an alarm is defined as a semicolon(;) divided list set in *AlarmReceivers* property. The property format is:

```
ALARM_NAME_1:LIST;OF;RECEIVERS_1
ALARM_NAME_2:LIST;OF;RECEIVERS_2
```

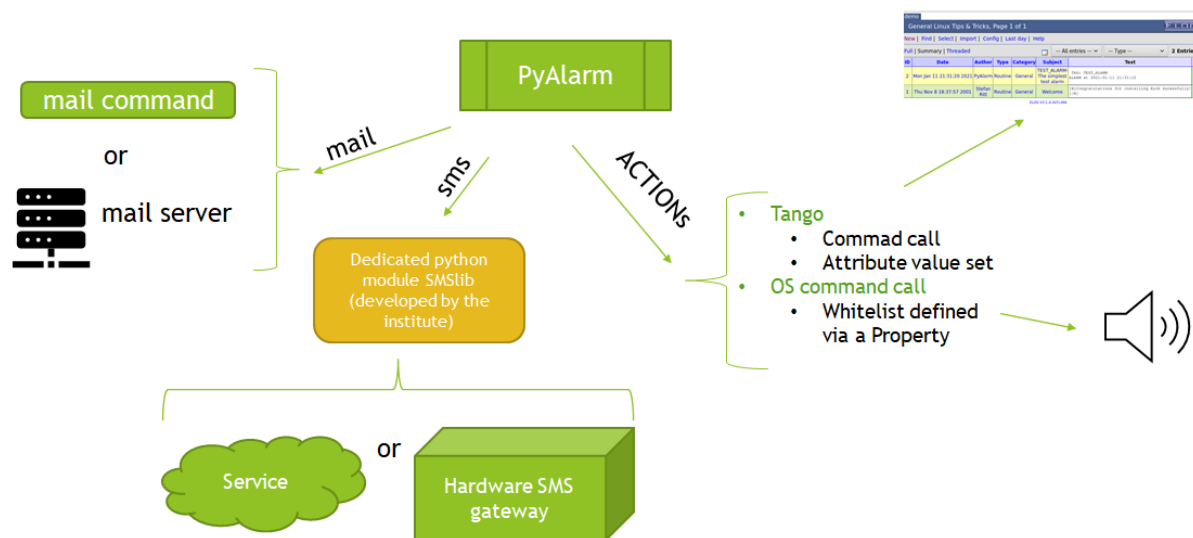


Fig. 3.1: Annunciation infrastructure

3.1 Actions

Actions (`ACTION (. . .)`) can be used to annunciate the alarm or to invoke some operations.

Actions can be invoked for the following events:

- **ALARM** - an alarm is triggered,
- **ACKNOWLEDGED** - an operator acknowledged the alarm,
- **RECOVERED** - the alarm condition become not active (formula evaluation become *False*),
- **REMINDER** - not acknowledged alarm reminder is being sent,
- **AUTORESET** - the alarm change automatically to *Normal* state (not active),
- **RESET** - an operator reset the alarm
- **DISABLED** - an operator disabled the alarm

For an arguments for command or variable set actions one can use the following *variables*:

- **\$ALARM/\$TAG/\$NAME** : Alarm name
- **\$DEVICE** : PyAlarm name
- **\$DESCRIPTION** : Description text
- **\$VALUES** : last values evaluated for that alarm
- **\$SNAP** : last values stored for that alarm
- **\$REPORT** : full report sent when the alarm was raised
- **\$DATE/\$DATETIME** : current time as YYYYMMDD_hhmm
- **\$MESSAGE** : type of alarm event (RESET,ALARM,REMINDER,...)
- **\$JSON** : all the previous fields in a JSON dictionary

These will be substituted with the value upon action invoking.

For operating system calls, list of commands to excute is limited to what is specified in the *AllowedActions* property.

Actions can be used to integrate external systems like a *TextTalker* or *Teams*.

3.2 PhoneBook

The Phonebook is defined as multientries *Phonebook* free property in *PANIC* section of free properties. Each line defines one named resceivers list: `%RECIVER_NAME: LIST;OF;RECEIVERS`, for example:

`%PIOTR GORYL:piotr.goryl@s2innovation.com;SMS:0048795794004`

Then, a `%RECIVER_NAME` can be used as a key in `AlarmReceivers` property.

3.3 Global Recivers

It is possible to define a default list of receivers/actions for a set of alarms. The set of alarms is defined by a regular expression. The global receivers list is defined in a *PyAlarm* class property:

GlobalReceivers: Receivers to be applied globally to all alarms. Declared as: `FILTER:receiver, ACTION (MESSAGE: . . .)`, for example:

```
*VC*:vacuum@cells.es,ACTION(RESET:command,t/t/t/stop)
```


NOTES ON ALARMS FORMULAS

Alarm conditions are provided as python like expressions, which are then calculated by the *TangoEval*. The *TangoEval* is an extended python *eval*.

Detail information and examples of formulas can be found in the PANIC documentation, here:

- [Alarm syntax recipes](#),
- [Custom alarms](#),
- [Example formulas](#),

4.1 Tango names resolving

The main enhancement to the standard python *eval* is the direct resolving of tango objects.

If one uses a string like the following (without quotes):

```
some/device/name{/attribute_name}{.value/all/time/quality/delta/exception}
```

(items in curly brackets are optional), the *TangoEval* resolve the string to:

- Tango *some/device/name* State, if the only device name is used,
- An attribute value indicated by *some/device/name/attribute_name*, if the string provides the *attribute_name*,
- An attribute property (*value/all/time/quality/delta/exception*), if it is provided in the string,

Please refer to [this documentation](#)

4.2 Special Keys/Macros

When providing formulas, the following special keys

- DEVICE: Returns PyAlarm device name
- DOMAIN, FAMILY, MEMBER: Parts of the device name
- ALARMS: Alarms managed by this device
- PANIC: API containing all declared alarms
- t: time since the device was started
- T(...): string to time
- str2time(...): string to time

- now, NOW(): current timestamp
- DEVICES: instantiated devices
- DEV(device): DeviceProxy(device)
- NAMES(*expression*): Finds all attributes matching the expression and return its names.
- **CACHE**: Saved values
- PREV: Previous values
- READ(attr): TangoEval.read_attribute(attr)
- **FIND**(*expression*): Finds all attributes matching the expression and return its values.
- **GROUP**(...)

For details, please look to the links to [Examples provided in the PANIC documentation](#)

4.2.1 Current timestamp - NOW()

Time: returns the epoch in seconds of the last value read

epoch is a date and time from which a computer measures system time

```
sys/tg_test/1/State.time < (now-60)
```

4.2.2 CACHE

This will trigger alarm if ALL values in the cache are equal, it is NOT the same as Delta because it checks only the first and last values:

```
not (lambda l:max(l)-min(l))([v.value for v in CACHE['elin/focus/b1coil/Position']])
```

4.2.3 String to time - T(...)

A temporal condition can be achieved using the T() macro in the formula. To re-enable it after a maintenance period:

```
T() < T('2020-01-09') AND (elin/v-rv/1/State != CLOSE)
```

4.2.4 FIND

```
((elin/focus/b1coil/Position > 55) OR (elin/focus/b2coil/Position > 55) OR elin/focus/  
↪b3coil/Position > 55) OR (elin/focus/b4coil/Position > 55))
```

is equal to:

```
any([s.value > 55 for s in FIND(elin/focus/b*coil/Position)])
```

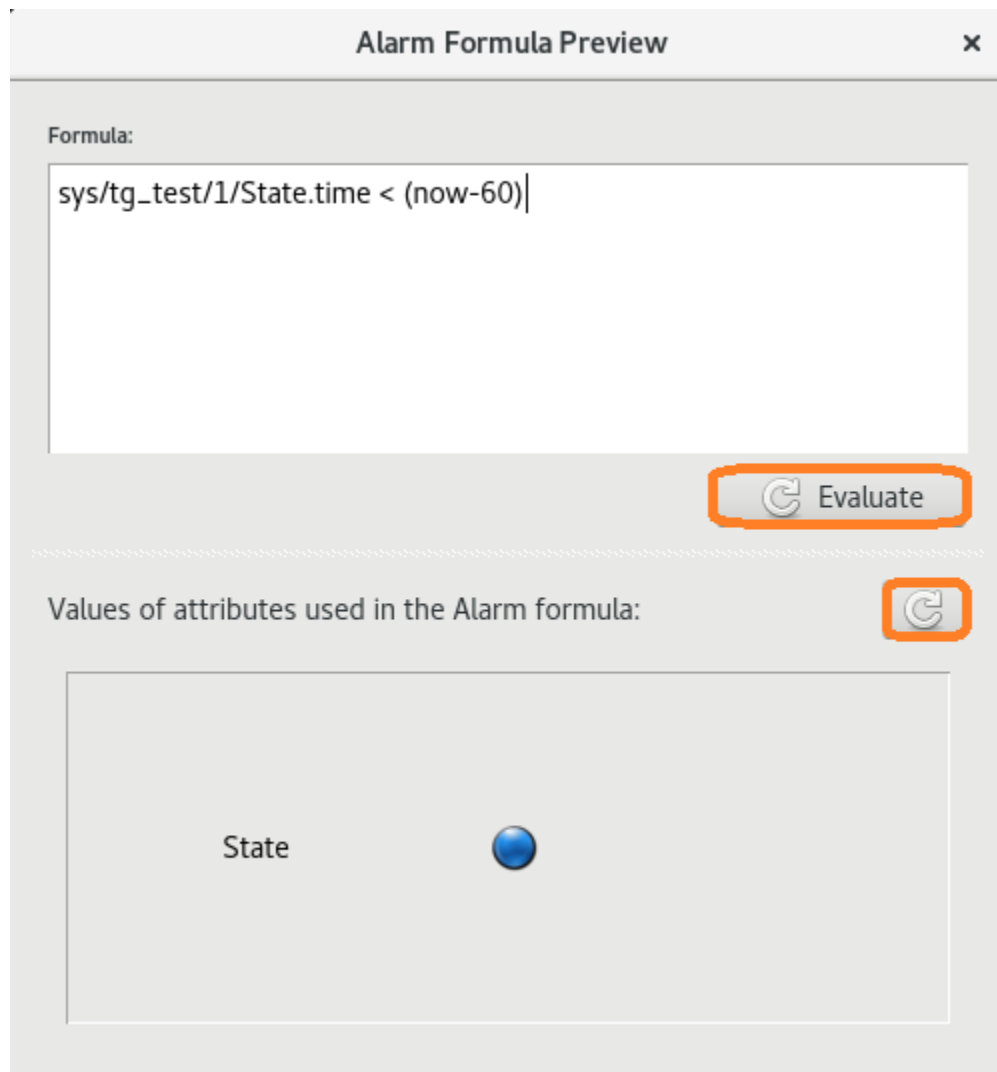


Fig. 4.1: NOW()

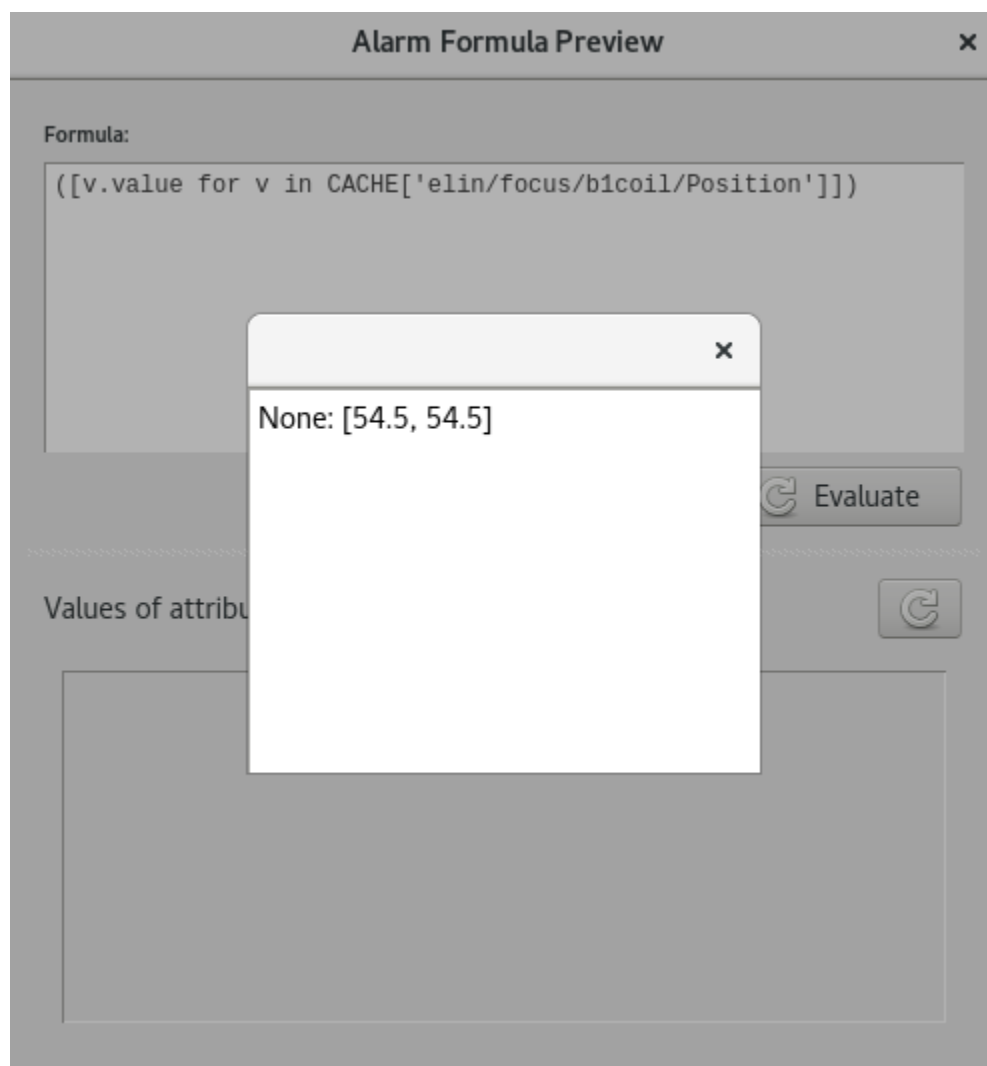


Fig. 4.2: CACHE

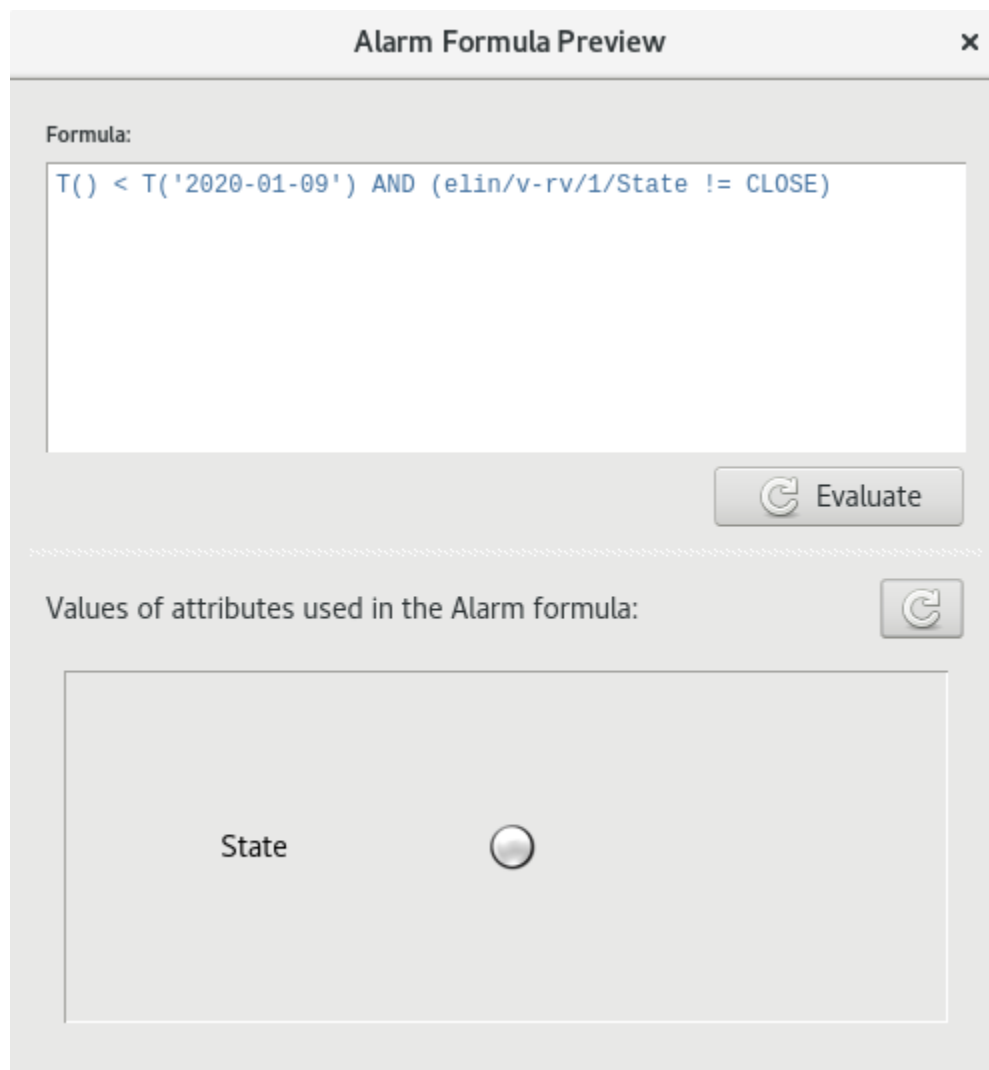


Fig. 4.3: T(...)

Alarm Formula Preview

Formula:

```
any([s.value > 55  
for s in FIND(elin/focus/b*coil/Position)])
```

Evaluate

Values of attributes used in the Alarm formula:

Position	54.50	0.00
Position	54.50	0.00
Position	54.50	0.00
Position	54.50	0.00

Fig. 4.4: FIND()

4.2.5 Grouping Alarms in Formulas

The proper way of grouping the alarms

```
ALARM_1: just/my/tango/attribute_1
ALARM_1: just/my/tango/attribute_2
```

then:

```
ALARM_1_OR_2: ALARM_1 or ALARM_2
```

Example:

```
((elin/v-rv/0/State != CLOSE) OR (elin/v-rv/1/State != CLOSE) OR (elin/v-rv/2/State !=
↪CLOSE) OR (elin/v-rv/3/State != CLOSE) OR (elin/v-rv/4/State != CLOSE) ) AND_
↪((elin/focus/b1coil/Position > 55) OR (elin/focus/b2coil/Position > 55) OR (elin/
↪focus/b3coil/Position > 55) OR (elin/focus/b4coil/Position > 55)))
```

is equal to:

```
VALVE_LINAC_TEST_ALARM:((elin/v-rv/0/State != CLOSE) OR (elin/v-rv/1/State != CLOSE)
↪OR (elin/v-rv/2/State != CLOSE) OR (elin/v-rv/3/State != CLOSE) OR (elin/v-rv/4/
↪State != CLOSE) )
```

```
Coil_Position_Linac_Alarm:((elin/focus/b1coil/Position > 55) OR (elin/focus/b2coil/
↪Position > 55) OR (elin/focus/b3coil/Position > 55) OR (elin/focus/b4coil/Position >
↪55))
```

```
(VALVE_LINAC_TEST_ALARM AND Coil_Position_Linac_Alarm)
```

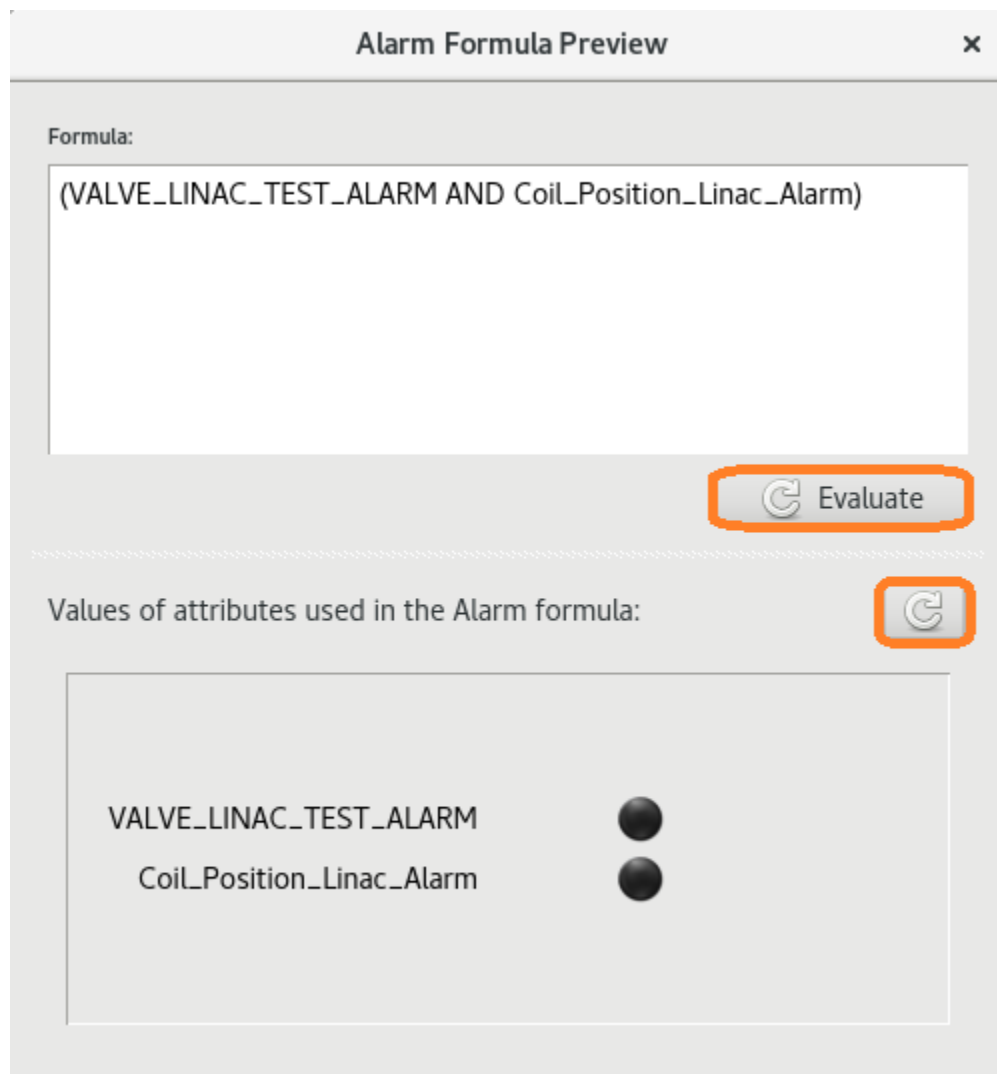


Fig. 4.5: Grouping

INTEGRATION WITH EXTERNAL SYSTEMS

Integration with external systems uses PANIC built-in annunciation capabilities:

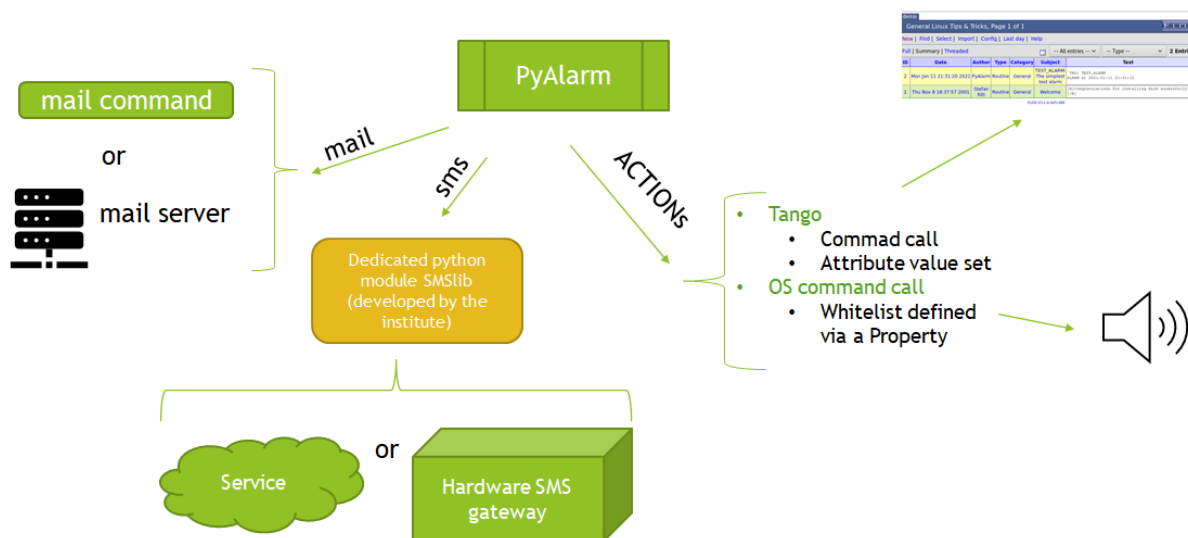


Fig. 5.1: Annunciation infrastructure

5.1 Mail

To enable PANIC to send emails, one needs to configure the following *PyAlarm* class properties:

5.1.1 MailMethod

It has to be set either to `mail` or `smtp[:host[:port]]`.

If it is set to `mail`, *PyAlarm* devices will call os command **mail** to send emails. This option requires that all machines where *PyAlarm* device servers are running have **mail** available and properly configured (with *sendmail* or *postfix*, for example).

If it is set to `smtp`, *PyAlarm* devices will send emails with use of a specified (*host*, *port*) SMTP server.

5.1.2 FromAddress

This property specifies sender address that will appear as Sender in mail and SMS.

5.1.3 MailDashOption

On some systems **mail** does not support default `-S` option. In that case, it is possible to require calling the **mail** with `-r` option to specify *from address*. One can do it by setting the *MailDashOption* property value to the sender address.

If the property is empty or not defined, `-r` is not used.

5.2 SMS

Sending SMS is possible with the use of an external hardware gateway or service. It requires an `smslib`. Then the configuration is provided with the following properties:

- **SMSConfig**: Arguments for `sendSMS` command (username and password), formatted as the following
username:password,
- **SMSMaxLength**: Maximum length of SMS messages,
- **SMSMaxPerDay**: Maximum SMS messages per day,

5.2.1 smslib

To enable sending of SMSs there should be a custom `smslib` library installed on all machines where *PyAlarm* device servers are running. This library needs to be provided/developed by the site using *PyAlarm*. An example library is available here: <https://github.com/S2Innovation/lib-s2i-smsdev>

The library shall be a python package (or module) named `smslib` defining (and exposing) a class called *SMSThread* which inherits from `threading.Thread`.

The *SMSThread* shall have a constructor with the following signature:

```
def __init__(self, message='', dest='', username='', password='', source='')
```

, a *PyAlarm* is initialising an *SMSThread* object with the following:

- **message** set to text to be sent,
- **dest** set to list of phone numbers,
- **username** and **password** parsed from the *SMSConfig* property,
- **source** set according to the *fromAddress* property,

After the *SMSThread* object is initialised, the *PyAlarm* is starting the thread, so sending of SMSs shall be done `def run(self):` method.

Note: The SMS sending could also be done with an *ACTION* receiver with use of an SMS sending device server or command/program.

Also the `smslib` may use a tango device to send an SMS. See the following modules used at SOLARIS:

- <https://github.com/S2Innovation/lib-s2i-smsdev>
- <https://github.com/S2Innovation/ds-s2i-smsgateway>

SOLARIS is using a Moxa GSM gateway, which provides an HTTP form to send SMSes.

5.3 A Text Talker

A text talker can be integrated with the use of an *ACTION* receiver.

Assuming that there is a text talker tango device *alarm/notifications/talker* with a command *talk* getting a text to speak as the *argin*, one can set up the following:

Phonebook entry:

```
%TALKER: ACTION(alarm:command,alarm/notifications/talker/talk, "There is an alarm " +
↪$DESCRIPTION)
```

Then adding %TALKER to an alarm annunciators/receivers list or to the GlobalReceivers property, will make the talker speak a message when an alarm triggers.

5.4 eLog or other web-based systems

To send notifications or to create an entry in a web-based system, one can use a tango command *ACTION* with a tango device in the middle.

As the web applications may have a limit on the number of concurrent connections or entries, it is recommended to buffer/filter the entries in a device server to prevent flooding the app in case of many alarms are triggered.

5.4.1 PSI eLog integration example

In addition to server-side eLog daemon, PSI elog provides a command-line tool called **elog**, which enables adding eLog entries from a command-line. It uses a set of command-line arguments to specify host, port, username and password to determine how the *eLog* can be accessed as well as a logbook name, entry's tags and content.

The **elog** command is used by an *ELogSender* tango device to send entries to eLog server.

```
elog -h <hostname> [-p port] [-d subdir]
                                Location where elogd is running
-l logbook                      Name of logbook
-s                              Use SSL for communication
[-v]                           For verbose output
[-w password]                  Write password defined on server
[-u username password]         User name and password
[-f <attachment>]              Up to 50 attachments
-a <attribute>=<value>         Up to 50 attributes
[-r <id>]                      Reply to existing message
[-q]                           Quote original text on reply
[-e <id>]                      Replace existing message
[-x]                           Suppress email notification
[-n 0|1|2]                    Encoding: 0:ELcode,1:plain,2:HTML
-m <textfile>] | <text>
```

More info is available at [PSI eLog webpage](#)

The *ELogSender* device has a command *create_entry* used by an *ACTION* annunciator to put an entry into a queue of entries to be sent to eLog. The queued entries are sent to the *eLog* at a constant interval. The queue size is limited to defined (i.e. 20) number of entries. If the queue is full, any new *create_entry* calls are discarded (a number of discarded entries are registered). This mechanism prevents the eLog server from overload in case of alarms' flood (multiple alarms triggered in a short time).

A way how the entry is created from a *create_entry argin* is defined by *ELogSender* device properties. These properties provide matching between elements of *argin* (which is *DevVarString*) and eLog fields (called attributes).

An example configuration may look like the following:

ELogSender class properties:

- **ArgumentParsers:** . (this provides additional variables to be used in eelog message, experimental),
- **ELogAdditionalArgs:** (this can be set to `-u pylaram eelog_password` to provide pyalarm credentials to eLog),
- **ELogCommand:** `/usr/local/bin/eelog` (where to find the **eelog** command),
- **ELogHost:** `localhost` (a hostname or IP where the *eLog* server is running),
- **ELogPath:** (an URL path of the *eLog*),
- **ELogPort:** `8080` (a tcp port of the *eLog*),
- **MaxQueueMessage:** `There is/are {%NumberOfRejectedEntries%} alarm(s) skipped to avoid flooding. {%n%}` (this provides a way to include information about rejected entries in a logbook entry),
- **MaxQueueSize:** `10` (entries que limit, if the que contains *MaxQueueSize* of entries, new one will be discarded, not sent to the *eLog* server),

alarm/ctl/elogsnd1 device properties:

- **EntryAttributes:** `Type={%3%}, Category={%4%}, Subject={%1%}-{%2%}, Author=PyAlarm`
- **EntryMessage:** `{%MaxQueueMessage%} {%0%}`
- **LogbookName:** `demo`

PANIC/PhoneBook free property entry: `%LOGBOOK:ACTION(alarm:command,alarm/ctl/elogsnd1/create_entry,$REPORT,$NAME,$DESCRIPTION,'Routine','General')`

PyAlarm/GlobalReceivers class property entry: `*:piotr.goryl@s2innovation.com,%LOGBOOK`

This will result in with entries looking as follows:

5.5 Knowledge database

A knowledge database link is configured with the *AlarmWikiLink* PyAlarm class property:

AlarmWikiLink: An URL to a WiKi page, where one can find more info on alarms. If set it will appear on the *AlarmEditor* widget. The URL may contain a key `{%ALARM%}` which is then substituted with an alarm tag. Example: `http://wiki.cps.uj.edu.pl/alarms/{%ALARM%}`,

demo

General Linux Tips & Tricks

[List](#) | [New](#) | [Edit](#) | [Delete](#) | [Reply](#) | [Duplicate](#) | [Find](#) | [Config](#) | [Help](#)

Message ID: **33** Entry time: **Wed Jan 13 00:09:40 2021**

Author:	PyAlarm
Type:	Routine
Category:	General
Subject:	MODULATORS_SYSTEM_FAULT-Gun and HV suply state mismatch

TAG: MODULATORS_SYSTEM_FAULT
ALARM at 2021-01-13 00:09:37

Alarm active since 2021-01-13 00:09:37
AlarmDevice: panic/example/pyalarm_standard
Description: Gun and HV suply state mismatch
Severity: ALARM
Formula: (elin/mod1/hv !=ON and elin/mod1/run == ON) or (elin/mod2/hv !=ON and elin/mod2/run == ON)

Values are:
elin/mod1/run: 0
elin/mod2/hv: 0
elin/mod2/run: 0
elin/mod1/hv: 1

Alarm receivers are:

MODULATORS_SYSTEM_FAULT ALARM
piotr.goryl@s2innovation.com

Fig. 5.2: An example eLog entry created from a PyAlarm

MAX-IV ELK STACK APPLICATION

For a description of an example installation of MAX-IV like deployment, see [the training VM documentation](#).

The integration with ELK stack let provide easy to configure visualisation dashboards as well as data analysis tools.

One can use it to provide info on KPIs like alarm's frequency, average, maximum and minimum response time, and so.

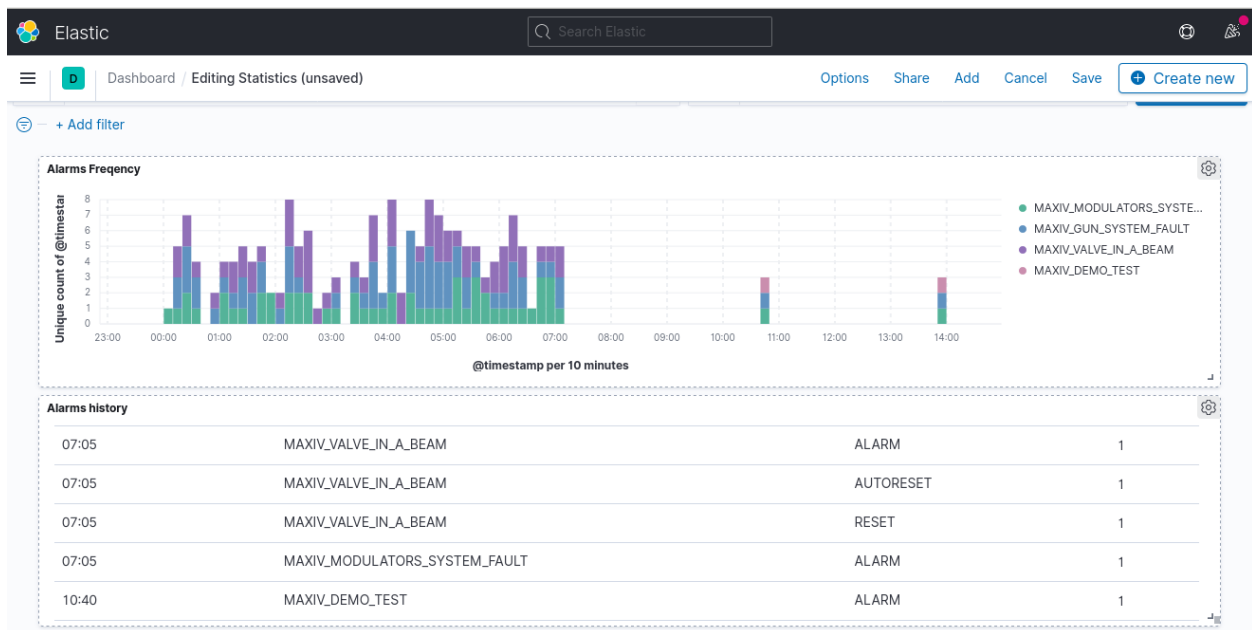


Fig. 6.1: An example Kibana dashboard showing alarm data

6.1 A demo of MAX-IV deployment

The demo is available on the training virtual machine. It is disabled from automatic startup to prevent resources consumption.

To start the demo, please run the script:

```
/home/panic/demo/maxiv/start.sh
```

The script starts the MAX-IV device server and services. The startup asks for *sudo* password.

After the services and MAX-IV PyAlarm is started, you can access the Kibana with a browser (i. e. **firefox**) on address: <http://training:5601>.

6.2 MAX-IV app description

MAX-IV uses its version of PyAlarm extended with a facility to send alarms' events to the Logstash/Elasticsearch. The DS is based on an old PyAlarm version (4.22.13).

Note: There is a related pull request sent by MAX-IV for PANIC mainstream repository. Due to in-meantime advance in PANIC development, the PR cannot be easily merged. It requires a bit of rewriting.

A MAX-IV device sends alarm information, packed into a *JSON* object to a Logstash server listening on the 5959 TCP port and address provided by its *LogStash* property.

The Logstash push the data to the Elasticsearch database according to its *configuration*.

The data available in the Elasticsearch database can be browsed with the Kibana web application.

The Kibana provides tools for data browsing, filtering, aggregating and visualising. Please refer to [Kibana documentation](#)

6.3 Deployment

To make the setup work one need to:

- install and configure [ELK stack](#):
 - Elasticsearch,
 - Logstash,
 - Kibana,
- provide a dedicated logstash *configuration file*,
- Install MAX-IV version of the PyAlarm device server, configure a PyAlarm device and set its **LogStash** property.

6.3.1 Logstash configuration example

To let PyAlarm send alarms to the Elasticsearch, a logstash configuration file (pipeline) has to be provided, so the Logstash accepts a *JSON* data on 5959 TCP port. The configuration file may look like the following:

```
input {
  tcp {
    port => 5959
    codec => json
  }
}
output {
  elasticsearch {
    hosts => ["localhost:9200"]
    index => "tango-alarms-%{+YYYY.MM.dd}"
  }
}
```

(continues on next page)

(continued from previous page)

```
        document_type => "alarm"  
    }  
}
```


PANIC SOURCE-CODE

PANIC is provided as a python package `panic`.

The package contains the following essential modules:

- `alarmapi.py`,
- `properties.py`,
- `ds/PyAlarm.py`,
- `gui/gui.py`,

and other supporting modules used by the above.

Below are a few notes about the source code.

7.1 `alarmapi.py`

The *alarmapi* module provides definitions of the following classes:

- *Alarm*, which keeps alarm states and let access to alarm attributes,
- *AlarmDS*, which provides an interface to manage PyAlarm devices,
- *AlarmAPI*, which gives access to Alarms and AlarmDS objects,

Both the PyAlarm device servers and PANIC GUI use the *AlarmAPI* to manage alarms.

Example of the usage of *AlarmAPI* is available in [the PANIC documentation](#).

7.2 `properties.py`

The module *properties.py* provides definitions of all Tango properties related to the PANIC system. It is an excellent place to start when looking for information about configuration properties.

Lists of properties defined in this module are used by *alarmapi*, *PyAlarm* and *GUI*.

7.3 PyAlarm.py

PyAlarm.py is the implementation of *PyAlarm* device server/class.

Below are important points about the structure of the sources:

- The source code starts with the import of modules. Optional modules (*PyTangoArchiving*, *smslib*) are loaded withing *try/except* clauses. If the *PyAlarm* cannot load a module (the module is not present), SNAP and/or SMS functionalities are set to disable.
- *AlarmHook* class is defined but not used.
- In addition to tango device interface definitions, there are the following important methods and objects:
 - *updateAlarms* is providing the main evaluation loop. It is started in a dedicated thread by *start* method,
 - *process_alarm*, where alarm state evaluation happen (call of *Eval* over a *formula*, state update). it is called by the above,
 - *send_alarm*, used by the above to annunciate a change of alarm state. The *send_alarm* uses other methods (*SendTelegram*, *SendSMS*, *trigger_snapshot*, *SendMail*, *SaveHtml*, *trigger_action*) to provide annunciations according to the configuration of the alarm,

In case of doubts or missing information on how a particular feature of the PANIC system work, it is worth to look into the source code of the methods mentioned above.

DEPLOYMENT AND CONFIGURATION

This section provides hints on deployment and *PyAlarm* devices configuration for SOLEIL.

8.1 PyAlarm dependencies

To start the PyAlarm the following python packages have to be installed:

- MySQL-python (this package is provided with `rpm sudo yum install MySQL-python`),

The rest of the packages may be installed with **pip** tool:

- numpy
- pytango
- fandango
- PyTangoArchiving
- taurus
- panic

, when installed with **pip**, it will install all required dependencies as well.

Note: CentOS 7 needs the EPEL repository to install the **pip** tool (`sudo yum install python-pip`).

On some systems (i.e. CentOS 7), it may be required to upgrade the default *pip* to the most recent version before installing the packages above:

```
sudo pip install --upgrade pip
sudo pip install --upgrade setuptools
```

8.2 PANIC GUI dependencies

The GUI requires few additional (in respect to PyAlarm) packages:

```
sudo yum install python-pyqt4
sudo yum install python-guiqwt
```

8.3 Number of devices/device servers

At SOLEIL, tango devices and device servers are grouped in subsystem-dedicate VMs.

It is then recommended to make PyAlarm device server available on all (virtual) machines either by providing python packages via shared NFS or local installation on the machines.

Then on all subsystem VM, there should be at least two PyAlarm devices.

It is suggested that a dedicate PyAlarm server instance provides each device (one device per server), a threading model of Python is not a real multi-thread.

The device server instance and devices naming conventions should take into account the purpose of the devices it serves (i. e. PyAlarm/rf_slow, for providing alarm/rf/slow_alarms device, which serves a *slow* alarms).

8.4 Timing configuration

As it was discussed during the training, it is recommended to have at least two PyAlarm devices per subsystem.

One will handle standard alarms (slow); the other will serve fast alarms.

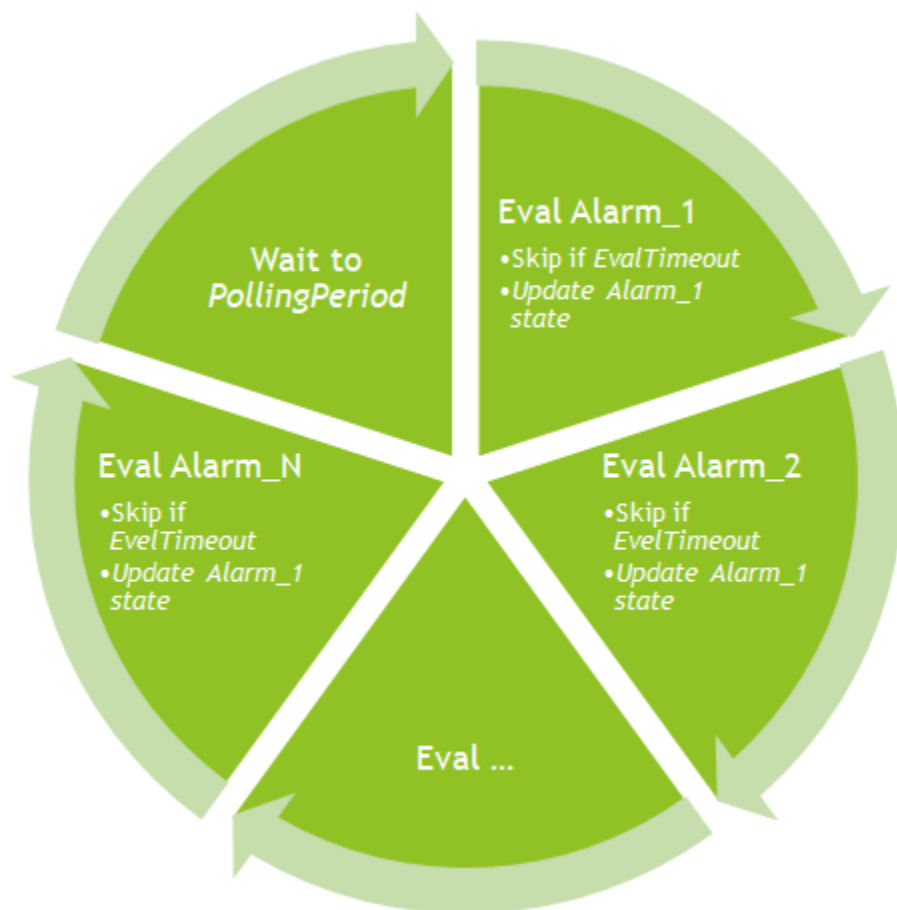


Fig. 8.1: Alarms evaluation cycle

8.4.1 Standard PyAlarm timing configuration

Most of the alarms will not require fast evaluations (are not based on short signals). For such alarms, the evaluation property may be set to 10 seconds or even more. The recommended settings are the following:

- **EvalTimeout:** 300, this will allow for slower devices,
- **AlarmThreshold:** 3, this will suppress transient communication problems,
- **PollingPeriod:** 10,
- **Reminder:** 3600,
- **AutoReset:** 3600, this may be adjusted according to control room practices. Having alarm auto-reset after one hour is a safe option if there is always an operator who looks from time to time on the alarm list dashboard, so none of the “mysteriously-self-healed” alarms is missed. If it is not the case, the *AutoReset* time shall be increased to make sure that none of the alarms will be missed without being noticed.

8.4.2 Fast PyAlarm timing configuration

PyAlarm devices providing alarms based on quickly changing values (some RF alarms) may need more often formulas evaluation to detect shorter/transient events.

- **EvalTimeout:** 10,
- **AlarmThreshold:** 1,
- **PollingPeriod:** 0.1,
- **Reminder:** 3600,
- **AutoReset:** 3600,

8.4.3 Startup

The alarms evaluation should start when all subsystems are already running. It is then recommended to start *PyAlarm* instances at *Run Level 5*.

Then, use the following property settings:

- **Enabled:** 300,
- **StartupDelay:** 120,

This way, the alarms will not be evaluated for two minutes preventing alarms from transient states. Then for 5 minutes, there will not be any notifications about alarms, except view on the PANIC GUI. Those settings will prevent notifications’ flood for operators during startup. Usually, during the startup operators are in the control room, and they have access to the *Panic GUI* to see activated alarms.

8.5 Exceptions handling configuration

At the beginning, it is recommended to set the exception related properties as follows:

- **RethrowState:** `True`,
- **RethrowAttribute:** `True`,
- **IgnoreExceptions:** `True`,

This will make sure that alarm will be triggered if there are issues with reading of any involved attribute.

8.6 UseTaurus

It is recommended to set the **UseTaurus** property to `True` for all *PyAlarm* devices. This will enable events for retrieving attributes values, so the formula's evaluation will not be slowed down by attributes reading.

8.7 Naming conventions

It is recommended to have a defined naming convention for:

- PyAlarm device servers instances
- PyAlarm device instances
- Alarms

The proposed instance name is: `PyAlarm/{subs}{#}`, where

- `{subs}` is an abbreviation of a subsystem to which the server is related (`rf`, `vac`, ...),
- `{#}` is one digit sequence number.

The suggested device naming schema is `alarm/{subs}/{standard/fast}{#}`, where:

- `{subs}` is an abbreviation of a subsystem to which the device relates (`rf`, `vac`, ...),
- `{standard/fast}` is one of `standard` or `fast` to denote the device timing configuration,
- `{#}` is one digit sequence Number.

Alarms' names may follow the following convention: `{SYS}_{SUBS}_{OBJECTORVALUE}_{ISSUE}`, i.w.: `SR_VAC_PRESSURE_DROP`, `I_RF_CAVITYTEMPERATUR_OVERFLOW`, `I_RF_MODULATOR_FAILURE`, `BL01_VAC_FRONTEND_CLOSED`

PANIC GUI

Table of Contents

- *PANIC GUI*
 - *PANIC GUI description and goals*
 - *GUI overview*
 - * *PhoneBook*
 - * *Trend*
 - * *Advanced configuration*
 - * *Alarm History Viewer*
 - * *Alarm Calculator*
 - *Attribute Finder*
 - * *Alarm Panel*
 - *Alarm edit/details panel*
 - *Configuration of the alarms list*
 - * *Sorting*
 - * *Filters*
 - * *Show active alarms*
 - *Context menu*
 - * *Alarm Details*
 - * *Preview Formula/Values*
 - * *View History*
 - * *Change Priority*
 - * *Acknowledge/Renounce Alarm*
 - * *Disable/Enable Alarm*
 - * *Edit Alarm*
 - * *Clone Alarm*
 - * *Delete Alarm*

- * *Advanced Config*
- * *TestDevice*
- *Alarms management*
 - * *Create new alarm*
- *Top bar menu*
 - * *Import and Export from CSV file*
 - * *Tools*

9.1 PANIC GUI description and goals

PANIC GUI allows the user to define configurations of the alarms. Those alarms can be saved in Tango Database and modified. PANIC GUI is gathering and managing information from all PyAlarm devices.

9.2 GUI overview

Application top bar has shortcuts to most popular options that are helpful in managing the alarms

9.2.1 PhoneBook

The Phonebook is tool to easy add, manage and remove receivers for alarms. Each line define one named receiver, for example:

```
%JAN KOWALSKI:jan.kowalski@s2innovation.com;0048123456789
```

After clicking on *PhoneBook* icon

User can see window to add, edit or remove receivers

After clicking on *Add* button user can add new receiver

9.2.2 Trend

After clicking on *Trend* icon

User can see Trend window of provided device attribute

9.2.3 Advanced configuration

After clicking on *Advanced Configuration* icon

User can select PyAlarm device and manage configuration

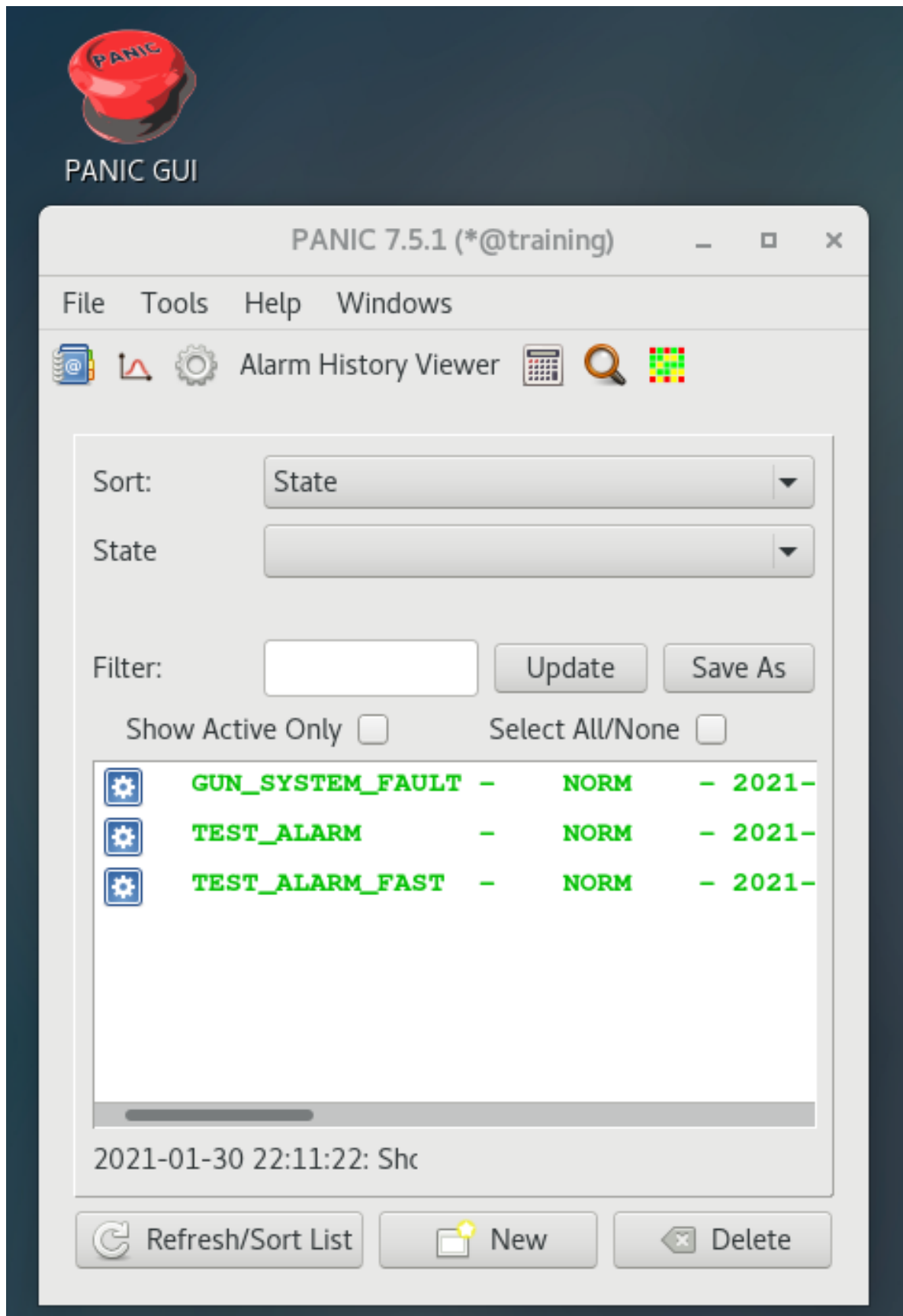


Fig. 9.1: Main view of PANIC GUI

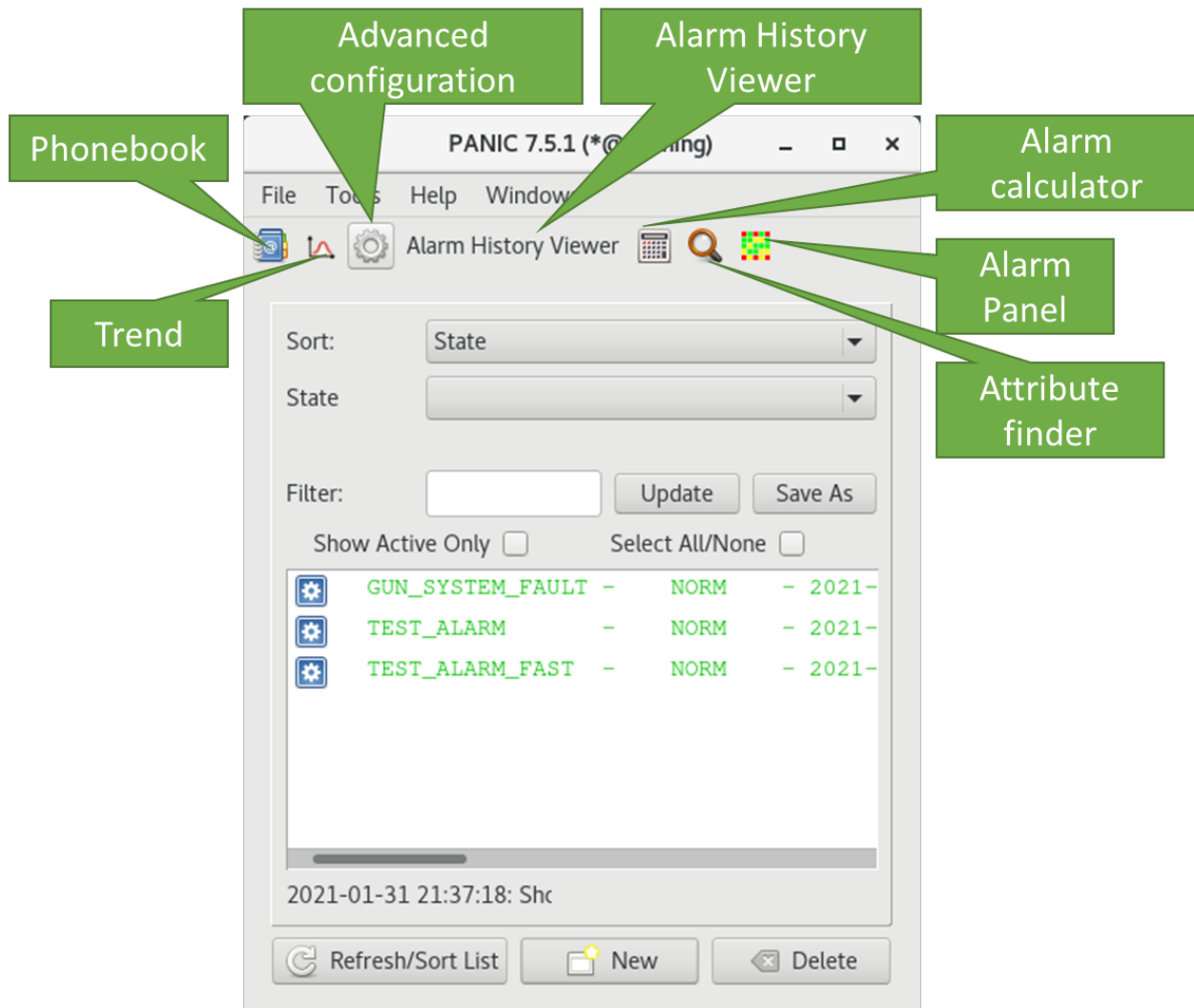


Fig. 9.2: PANIC GUI overview

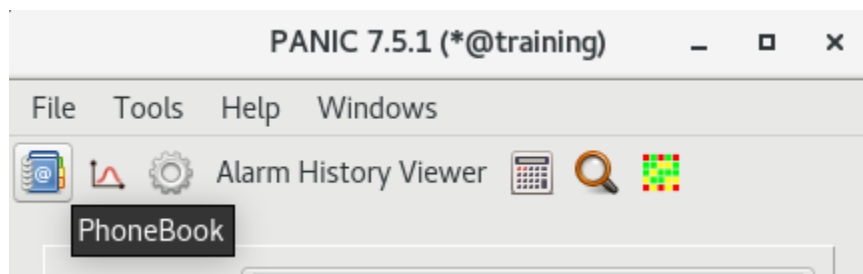


Fig. 9.3: PANIC GUI, PhoneBook

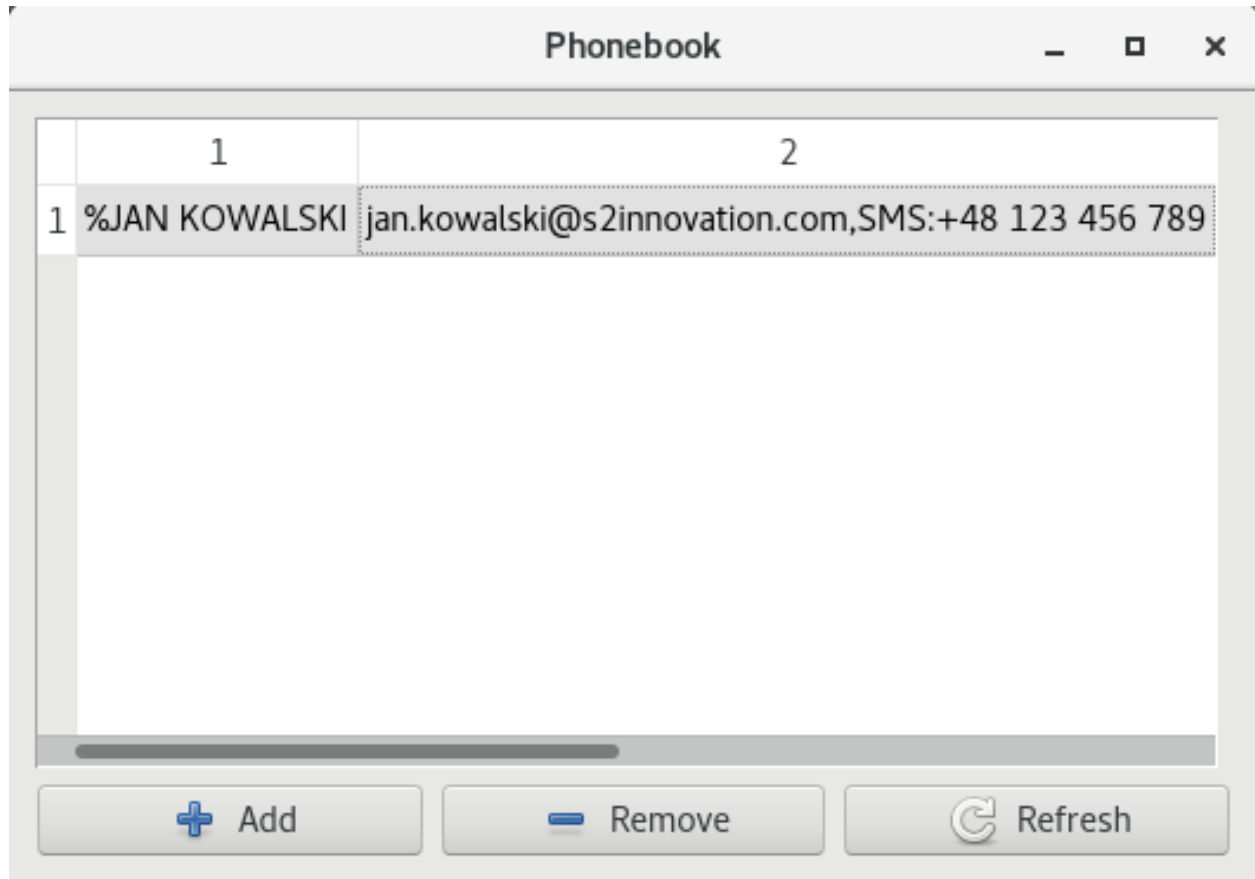


Fig. 9.4: Phonebook main window

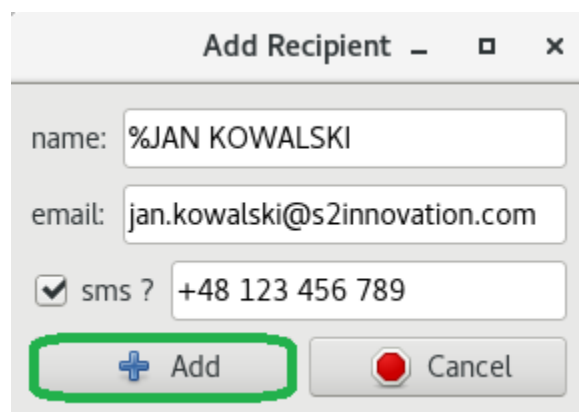


Fig. 9.5: Phonebook add resceiver

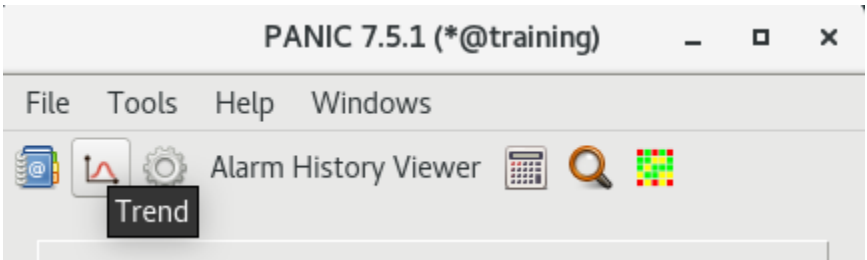


Fig. 9.6: PANIC GUI, Trend

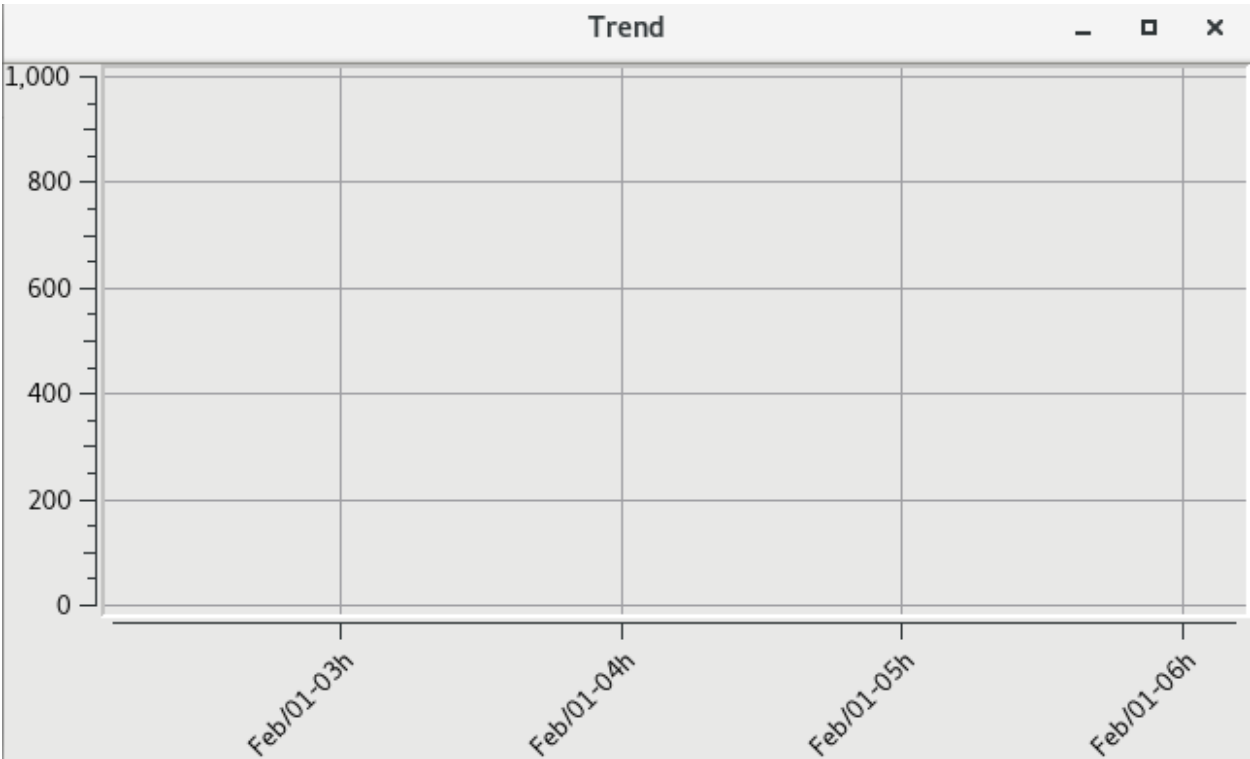


Fig. 9.7: Alarms trend main view

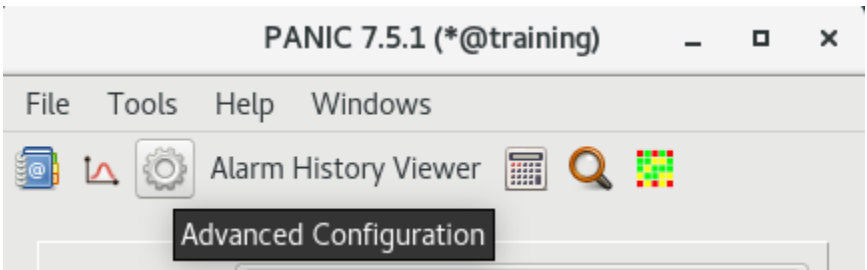


Fig. 9.8: ANIC GUI, Advanced Configuration

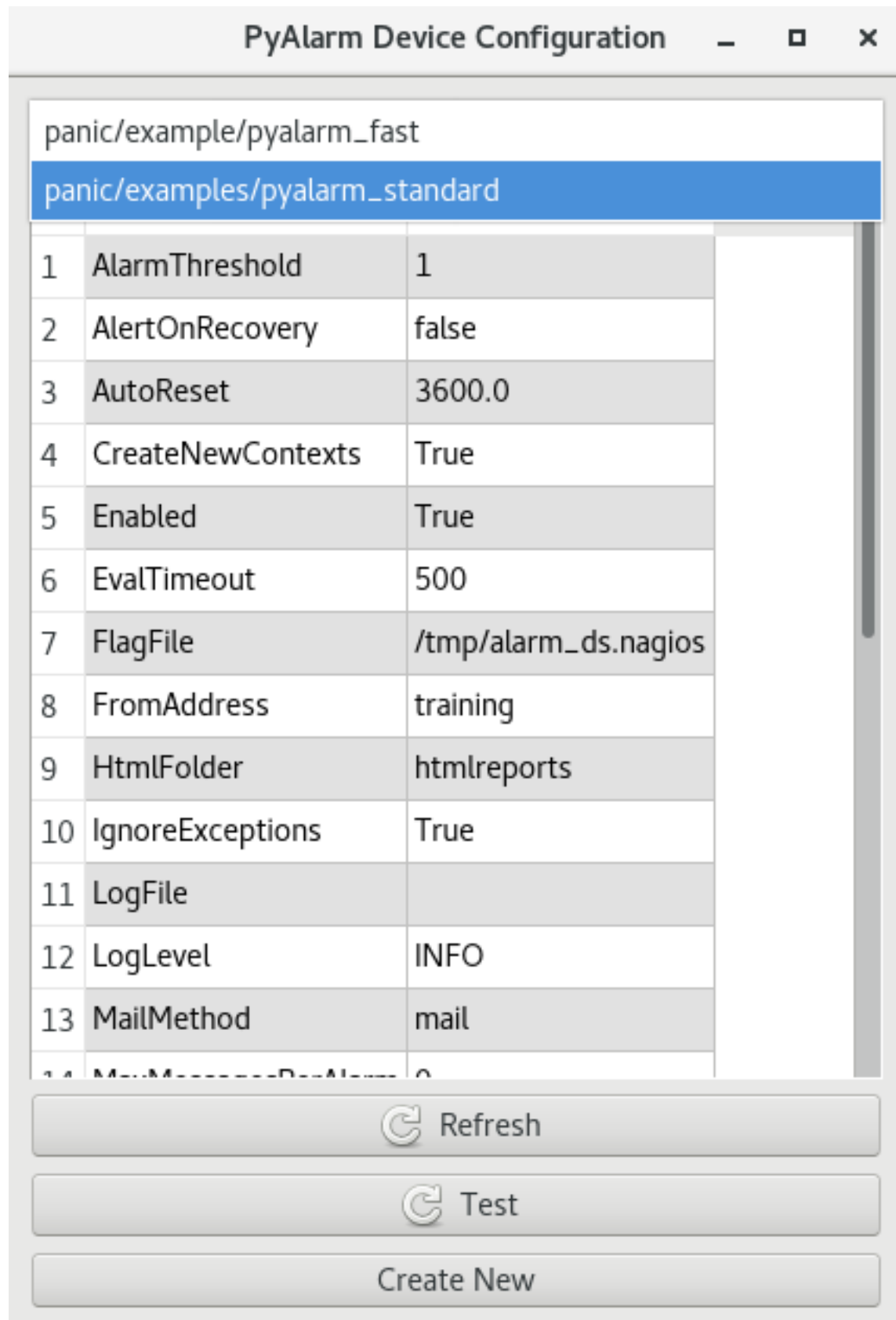


Fig. 9.9: PyAlarm Device Configuration

9.2.4 Alarm History Viewer

After clicking on *Alarm History Viewer* icon

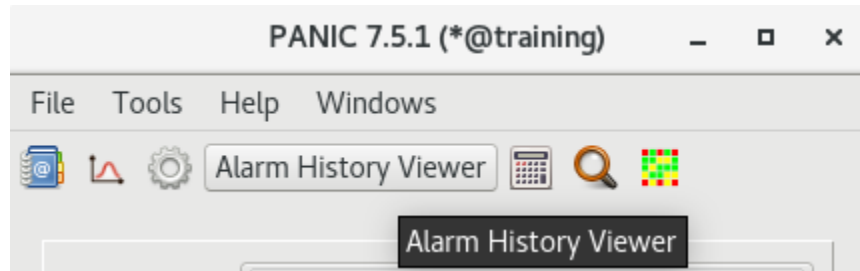


Fig. 9.10: PANIC GUI, Alarm History Viewer

User can see all available alarms history or select alarm and check history of chosen alarm.

After clicking on number in first column and next *Open Snapshot* button, user can see context of selected Snapshot.

9.2.5 Alarm Calculator

After clicking on *Alarm Calculator* icon

User can create and validate formula of the alarm.

After clicking on *Evaluate* button user can validate to verify that provided formula will trigger the alarm.

Attribute Finder

After clicking on *Attribute Finder* icon

User can search for devices and attributes using wildcards

After clicking on button in Archiving column user can verify that selected attribute is archived.

9.2.6 Alarm Panel

After clicking on *Alarm Panel* icon

User can open window showing state of configured alarms.

9.3 Alarm edit/details panel

After double-click on selected item on alarms list

User can check details of an alarm.

After clicking on *Edit* button user can edit selected alarm

User can e.g. modify formula of selected alarm

After clicking on *Evaluate* button user can verify provided formula

After clicking on *Save* button user can store modification.

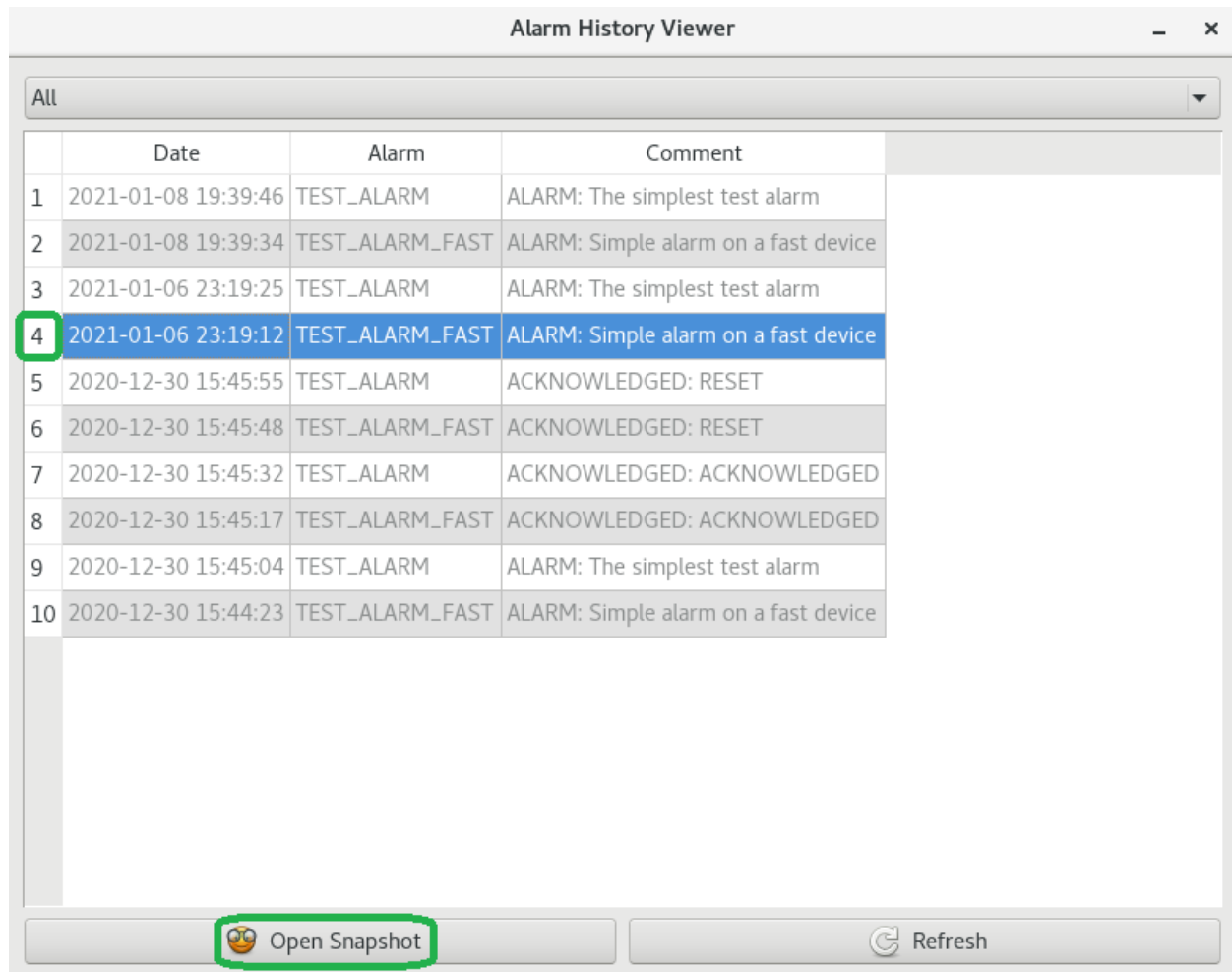


Fig. 9.11: Alarm History Viewer

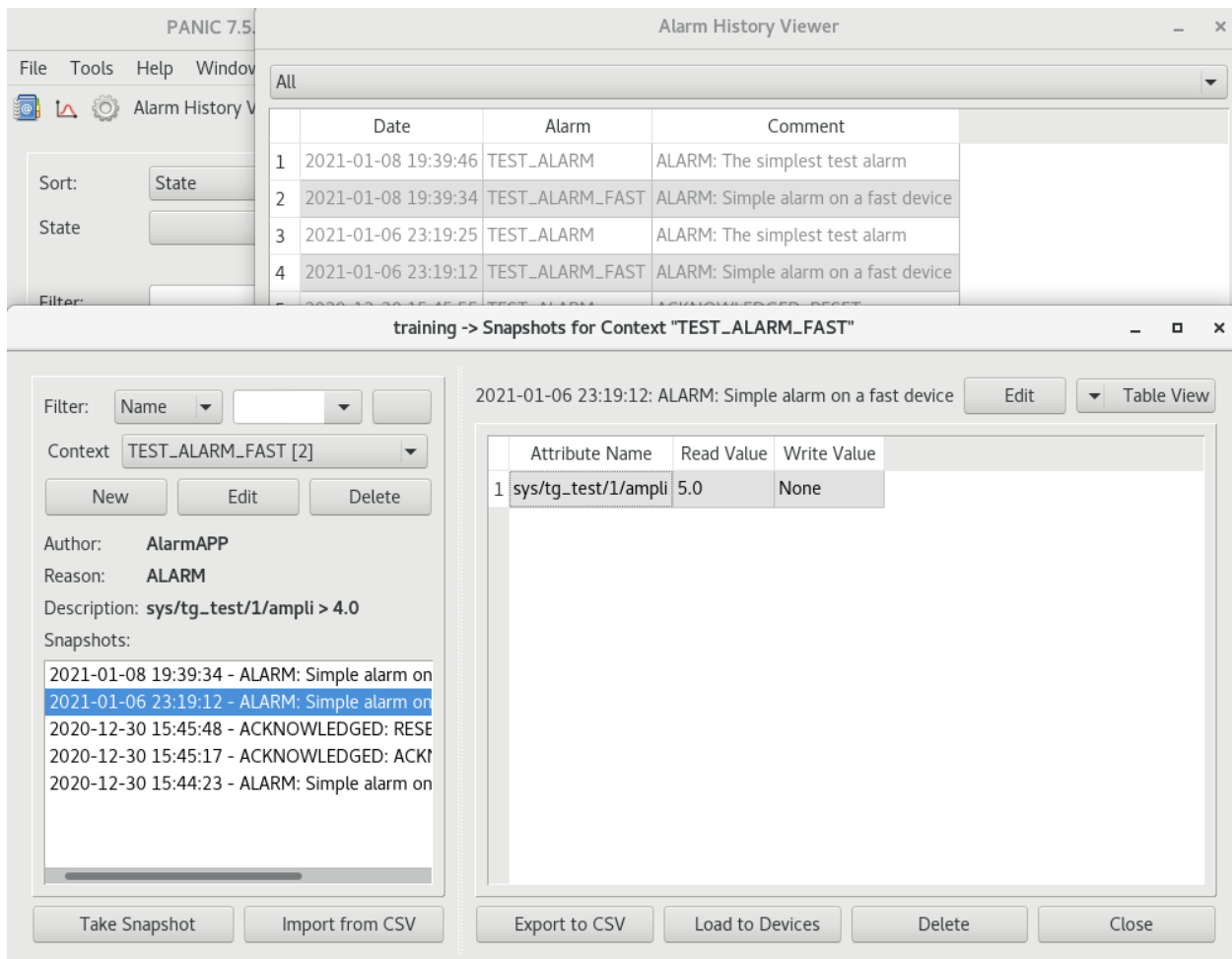


Fig. 9.12: Snapshot of selected alarm context

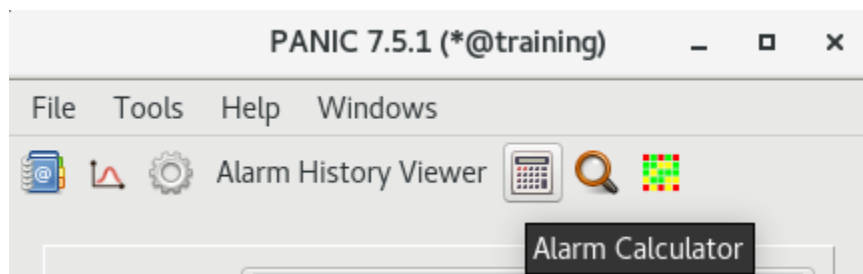



Fig. 9.13: PANIC GUI, Alarm Calculator


Alarm Formula Preview

Formula:


```
(sys/tg_test/1/State == RUNNING)
AND (sys/tg_test/1/ampli >= 3.0)
```

 Evaluate

Values of attributes used in the Alarm formula:



State



ampli

0.00

0.00

Fig. 9.14: Alarm Formula Preview

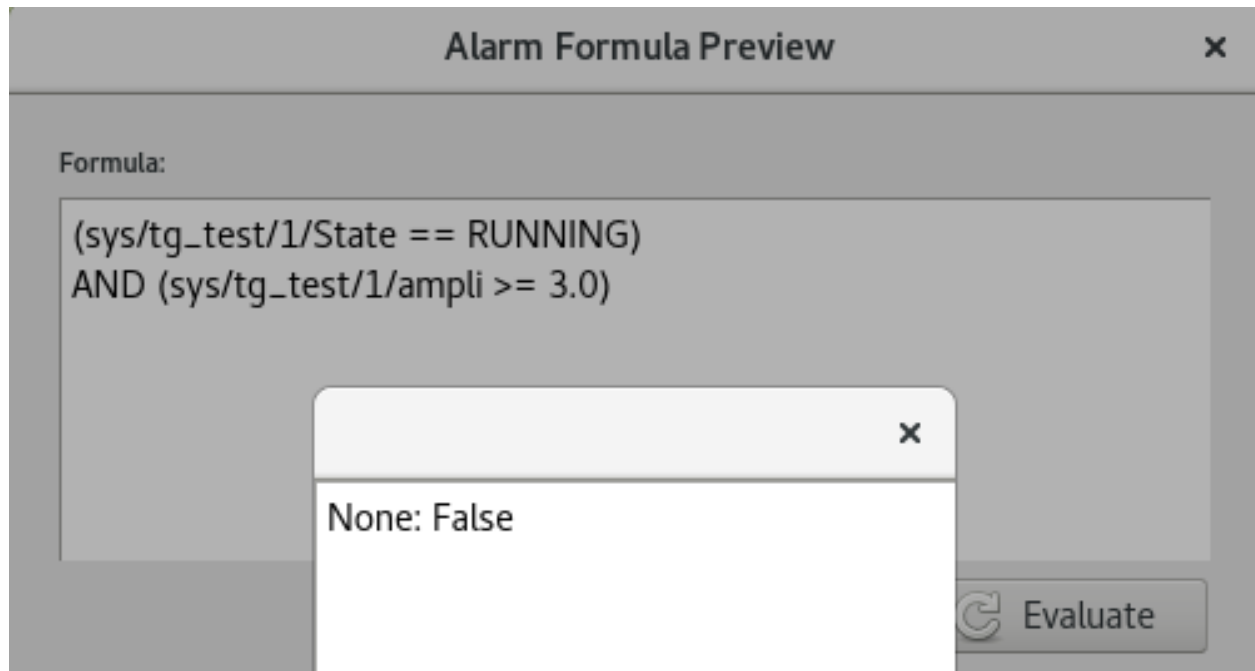


Fig. 9.15: Evaluate of alarm formula

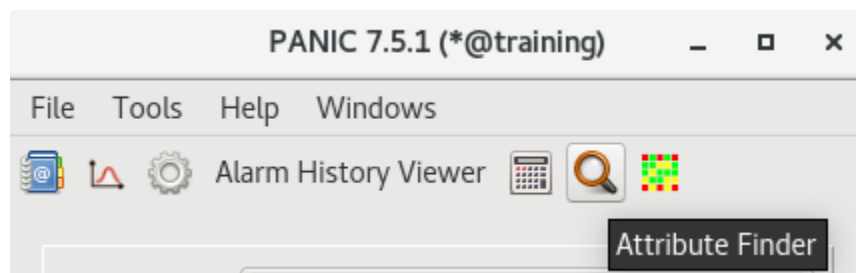


Fig. 9.16: PANIC GUI, Attribute Finder

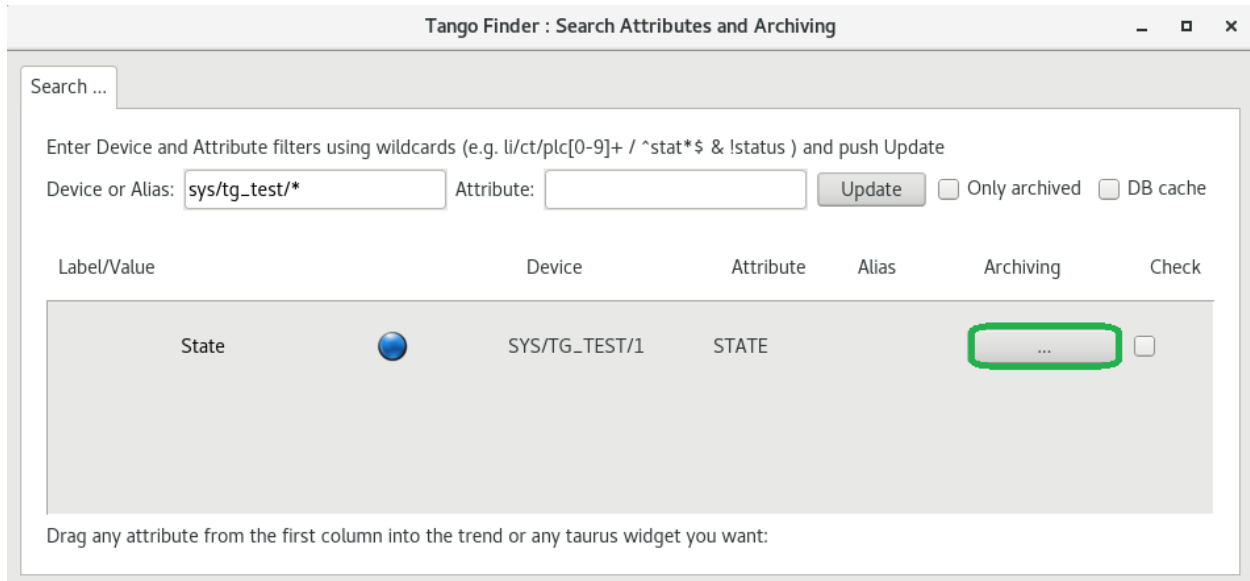


Fig. 9.17: Attribute Finder

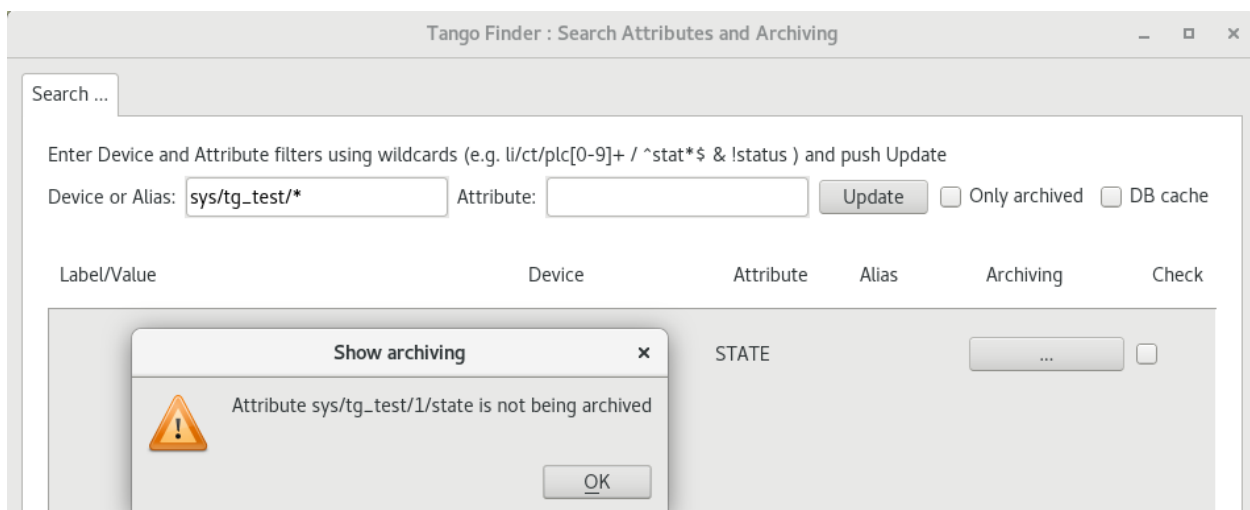


Fig. 9.18: Verify attributes archived

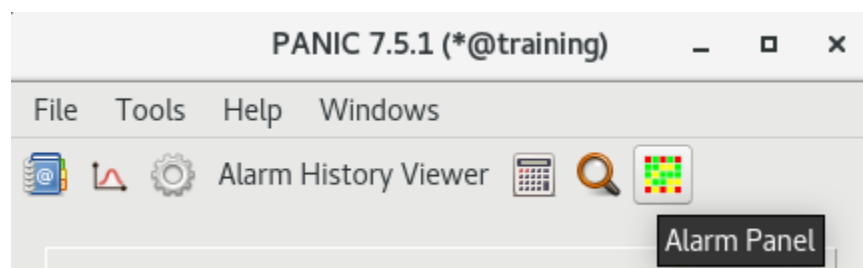


Fig. 9.19: PANIC GUI, Alarm Panel

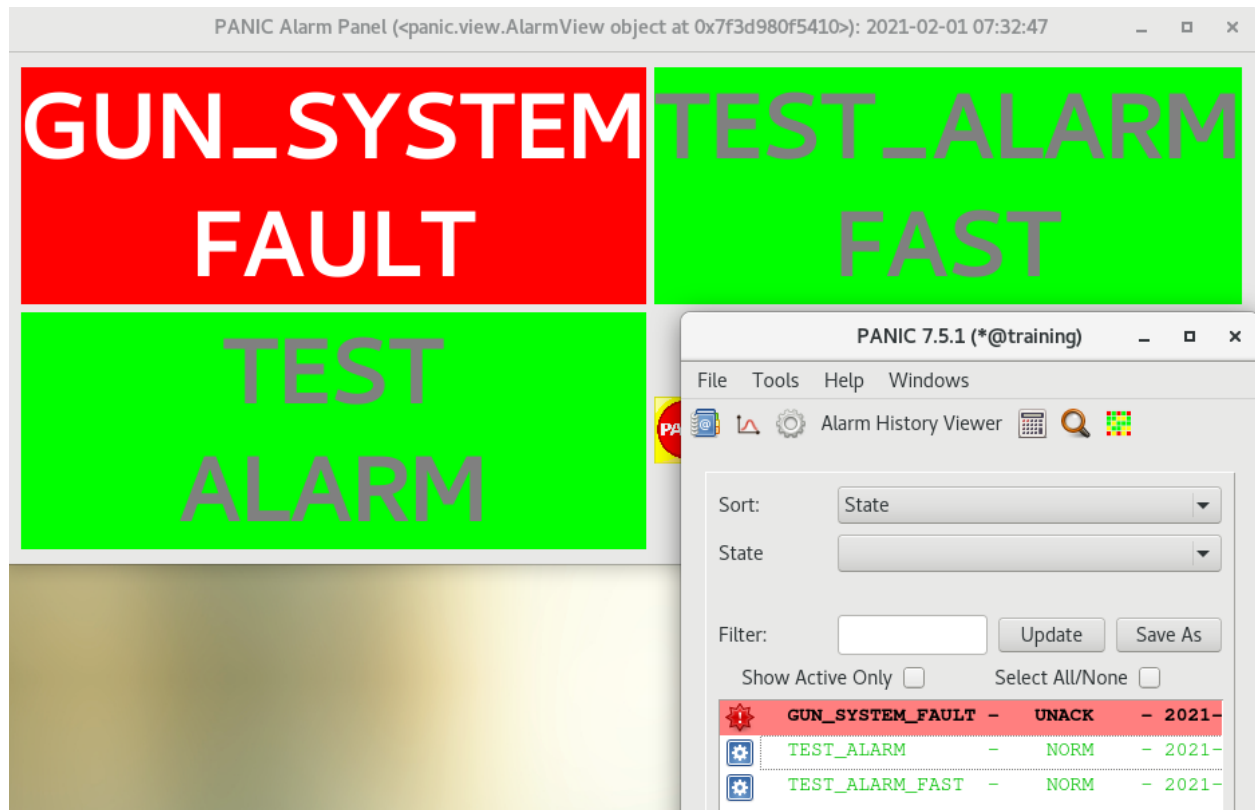


Fig. 9.20: PANIC Alarm Panel

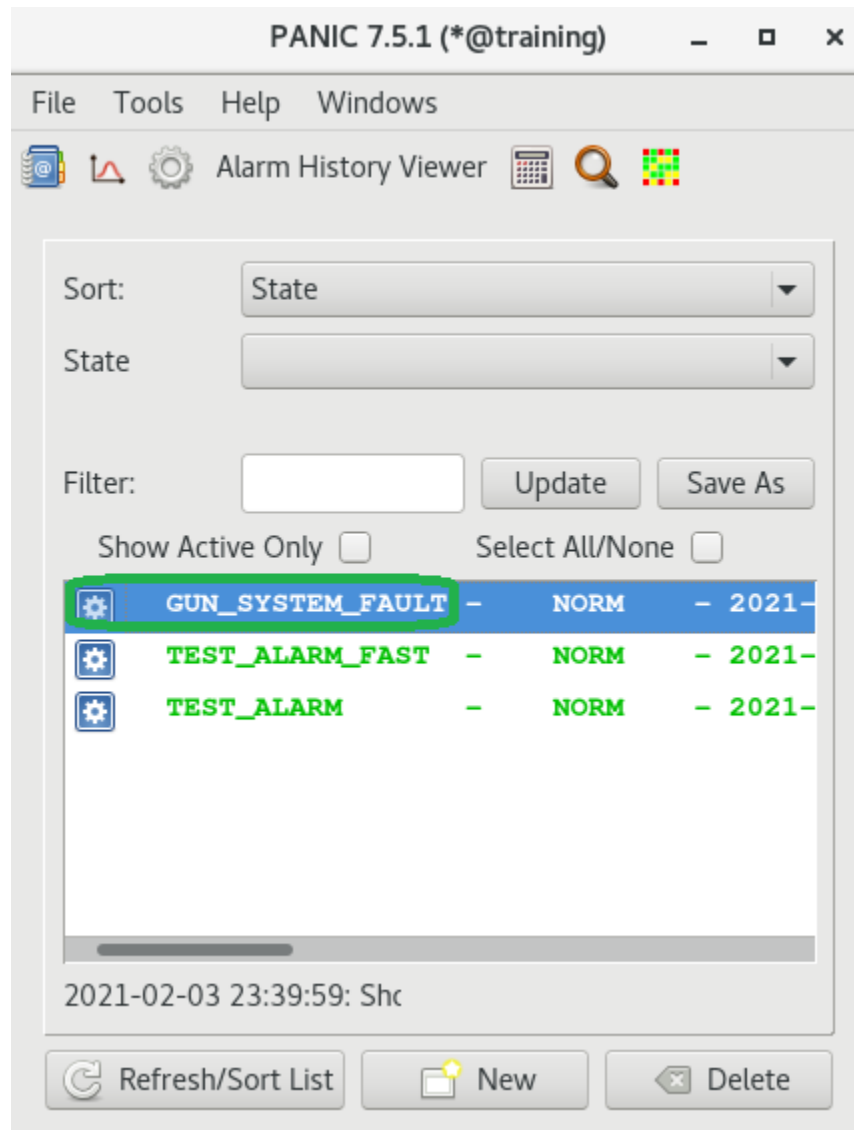


Fig. 9.21: Alarms list

ALARM: GUN_SYSTEM_FAULT ×

Tag:

State: NORM 🔍 Last Report ↶ Reset

Disabled: ☐

Acknowledged: ☐

Device: ⚙️

Priority:

Description:

Annunciators: +

Formula:

🔄 Evaluate

📝 Edit 👁️ Show Values 💾 Save ❌ Cancel

Fig. 9.22: PANIC GUI, view alarm details

ALARM: GUN_SYSTEM_FAULT ×

Tag:

State: NORM 🔍 Last Report ↶ Reset

Disabled: ☐

Acknowledged: ☐

Device: ⚙️

Priority:

Description:

Annunciators: +

Formula:

🔄 Evaluate

📝 Edit 👁️ Show Values 💾 Save ❌ Cancel

Fig. 9.23: Edit alarm

ALARM: GUN_SYSTEM_FAULT ×

Tag:

State: NORM 🔍 Last Report ↶ Reset

Disabled: ☐

Acknowledged: ☐

Device: ⚙️

Priority:

Description:

Annunciators: +

Formula:

```
((elin/gun/hv != ON)
and (elin/gun/hv/HighVoltage > 75))
and (elin/beam/run == ON)
```

🔄 Evaluate

📝 Edit 👁️ Show Values 💾 Save ❌ Cancel

Fig. 9.24: Modify formula

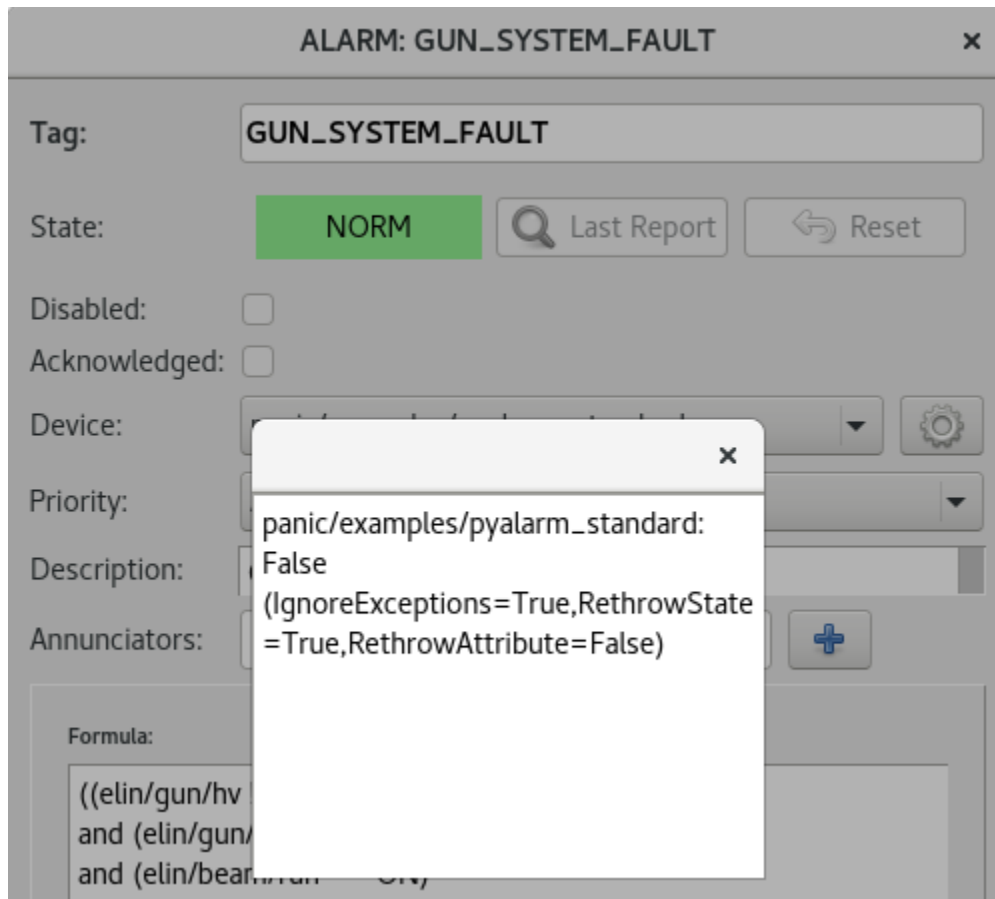


Fig. 9.25: Evaluate formula

9.4 Configuration of the alarms list

User can customize list of presented alarms list.

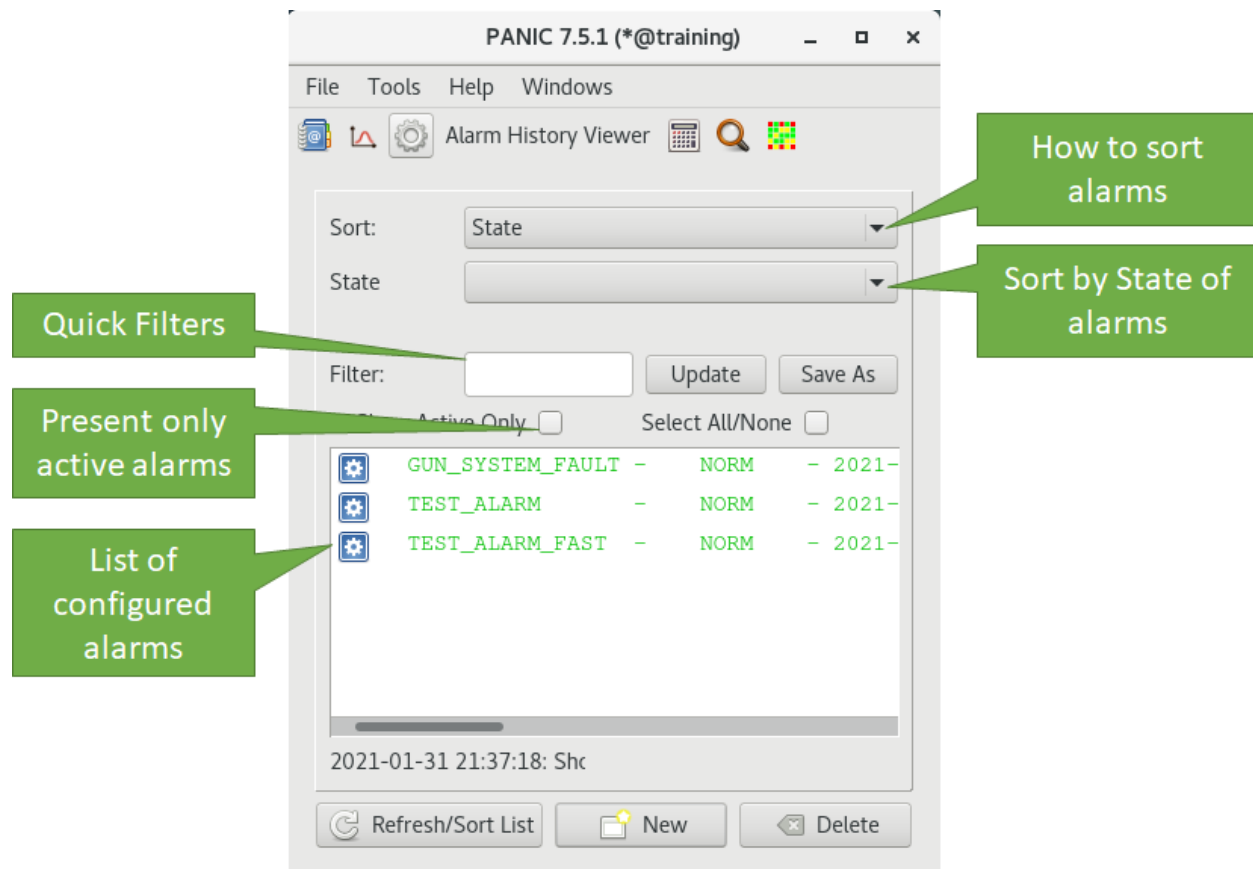


Fig. 9.26: Configuration of presented alarms

9.4.1 Sorting

After clicking on drop-down list

User can sort alarms by:

- State
- UserFilters
- Priority
- Devices
- PreCondition
- Annunciator
- Receivers
- Domain
- Family

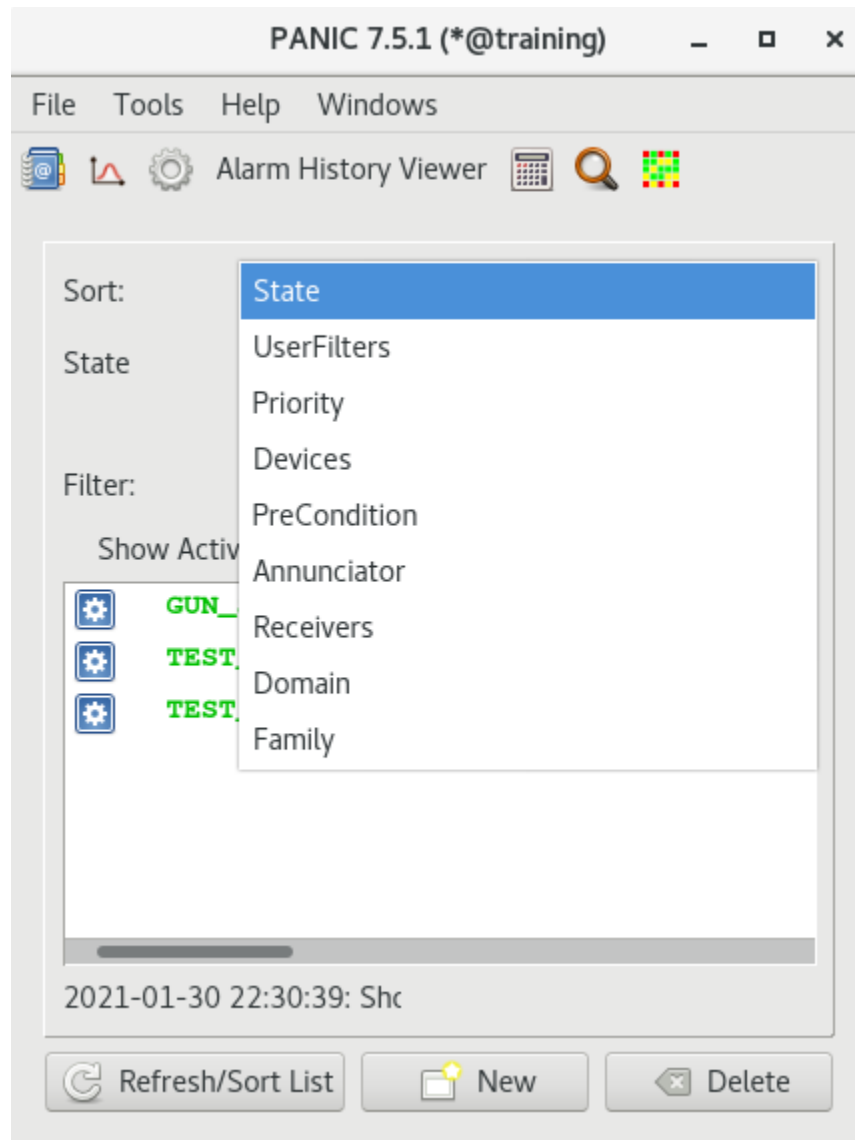


Fig. 9.27: Sorting alarms

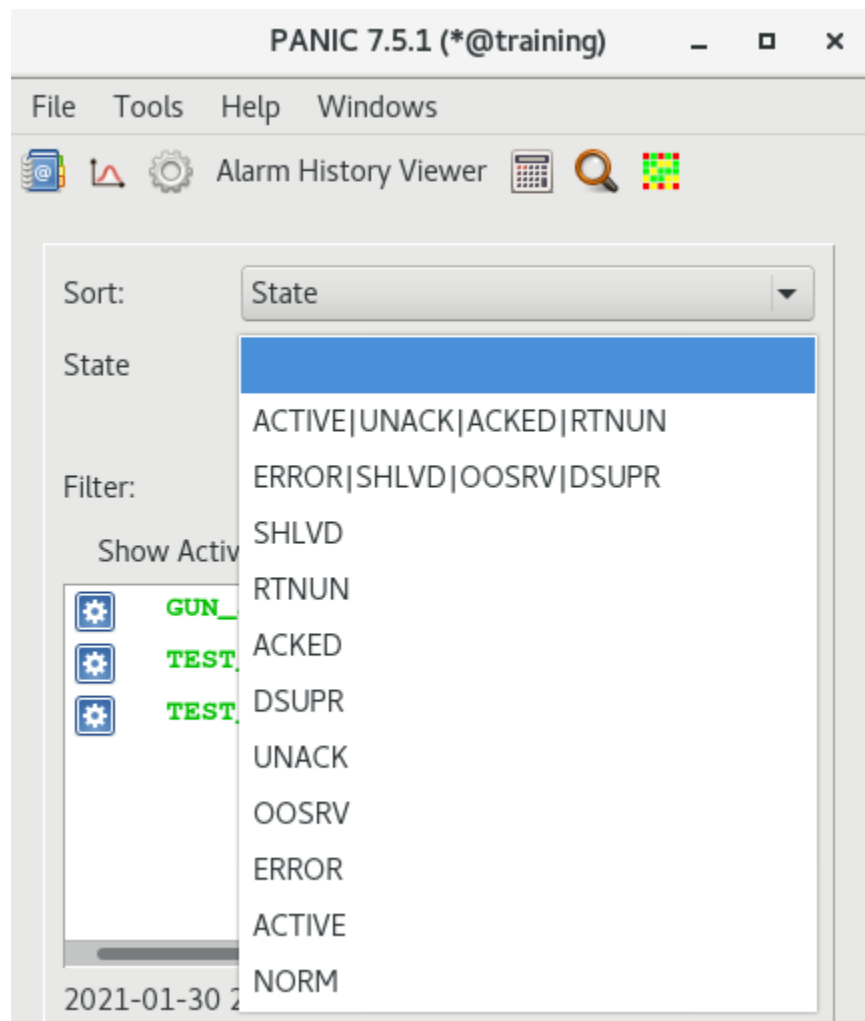


Fig. 9.28: State

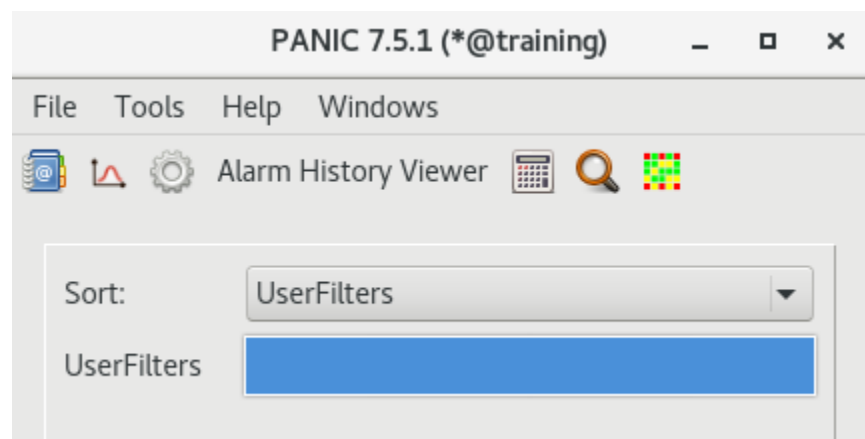


Fig. 9.29: UserFilters

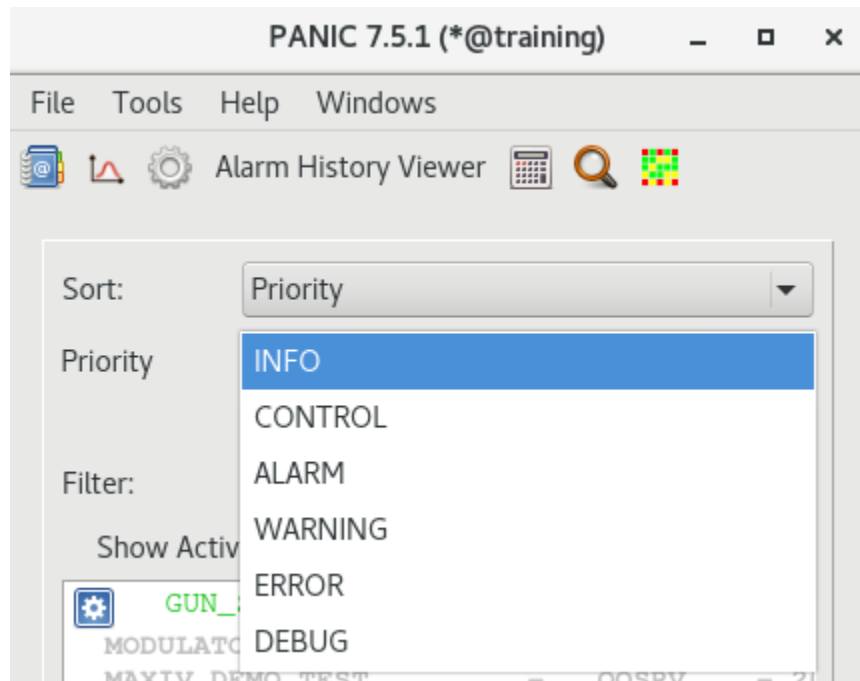


Fig. 9.30: Priority

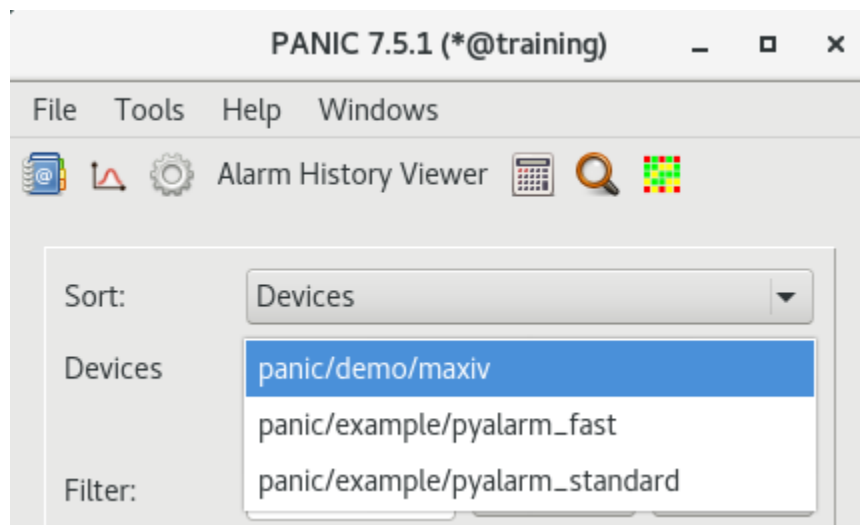


Fig. 9.31: Devices

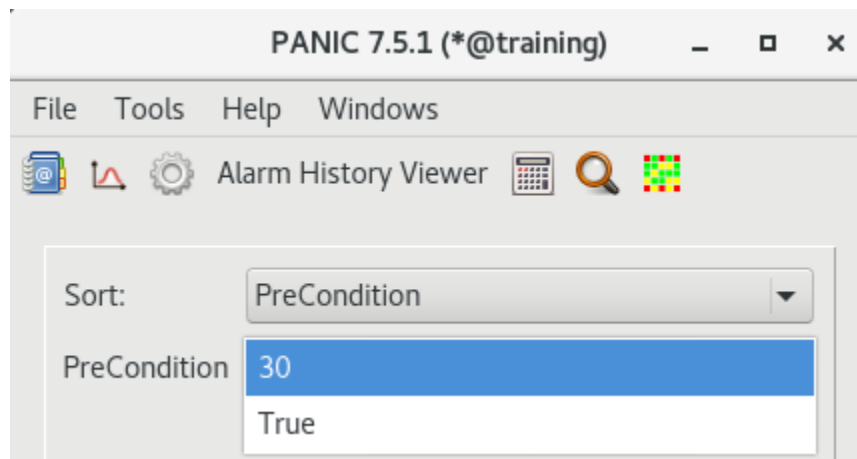


Fig. 9.32: PreCondition

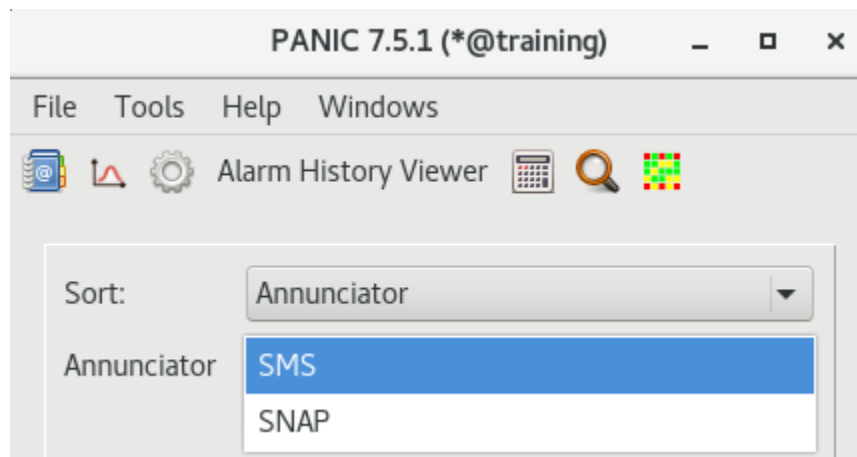


Fig. 9.33: Annunciator

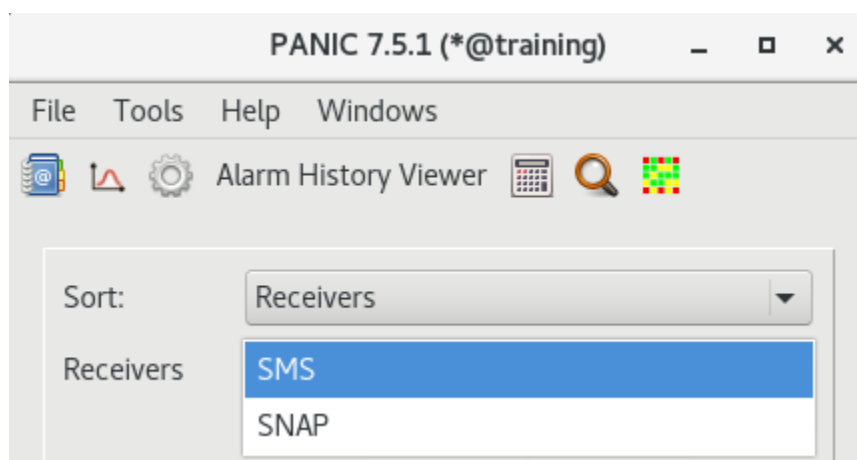


Fig. 9.34: Receivers

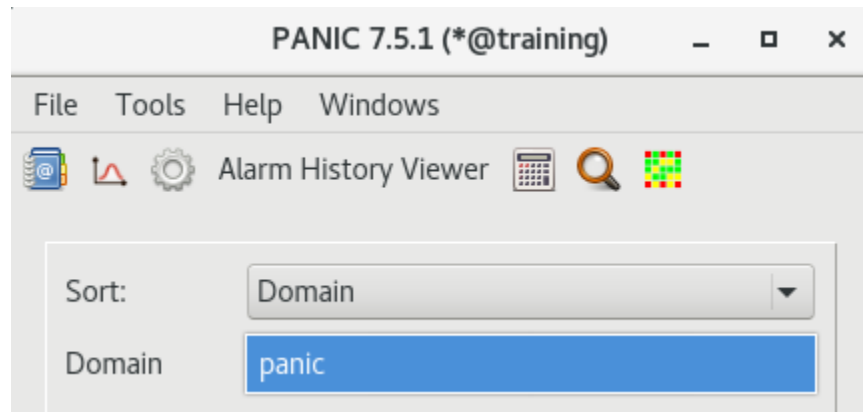


Fig. 9.35: Domain

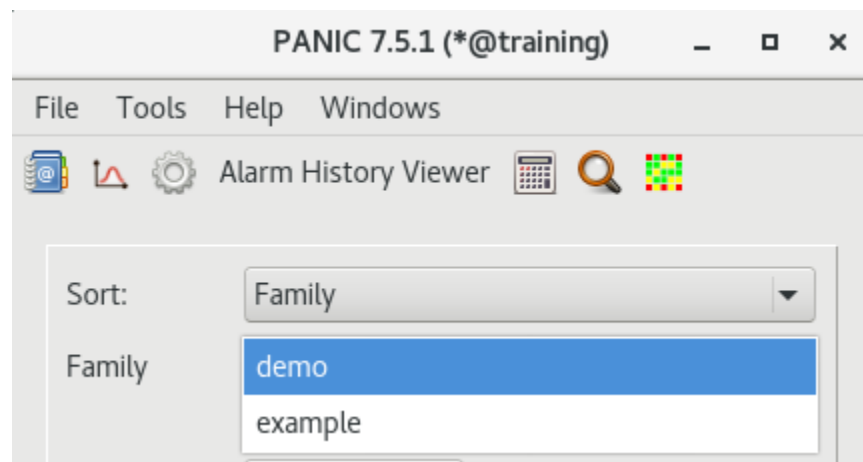


Fig. 9.36: Family

9.4.2 Filters

After clicking on filter text box:

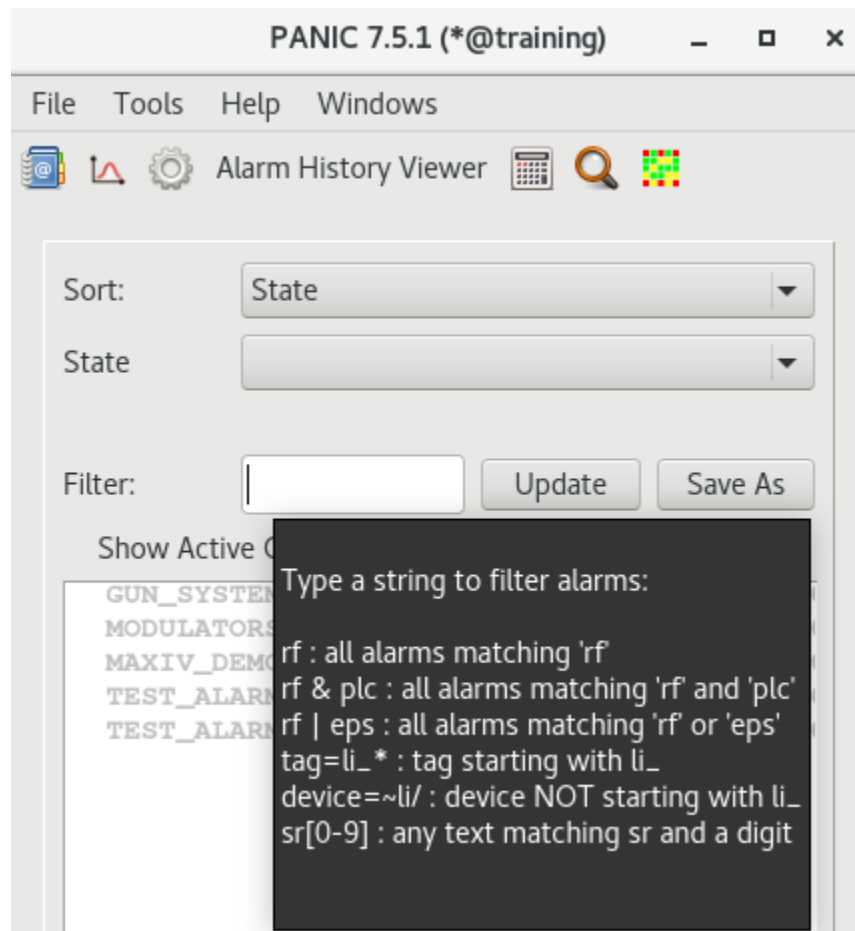


Fig. 9.37: Filters

User can create quick filter and save it.

9.4.3 Show active alarms

After clicking on *Show Active Only* checkbox.

User can see only active alarms.

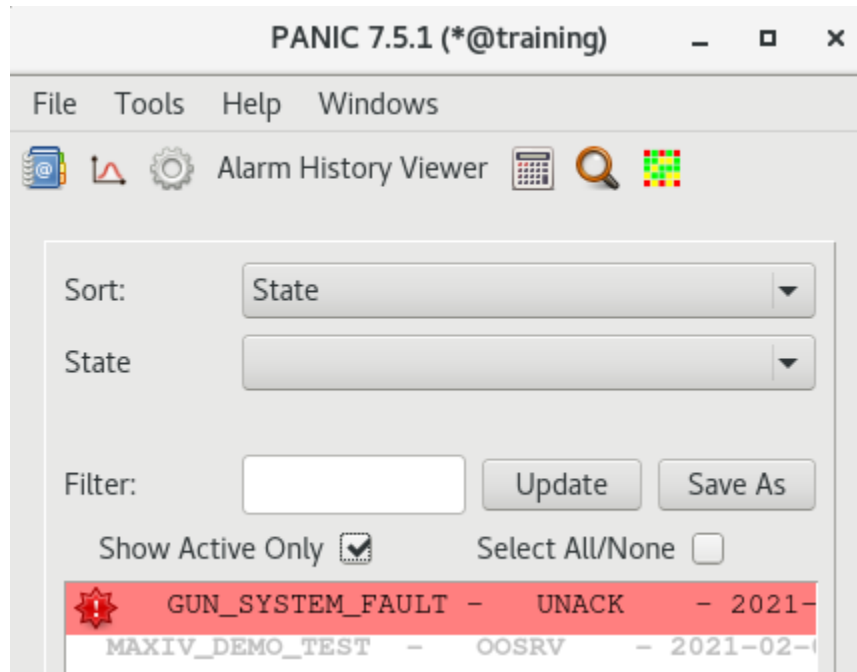


Fig. 9.38: Show active alarms

9.5 Context menu

After right click on selected alarm on the list

User can choose:

- See Alarm Details
- Preview Formula/Values
- View History
- Change Priority
- Reset Alarm(s)
- Acknowledge/Renounce Alarm(s)
- Disable/Enable Alarm(s)
- Edit Alarm
- Clone Alarm
- Delete Alarm
- Advanced Config
- TestDevice

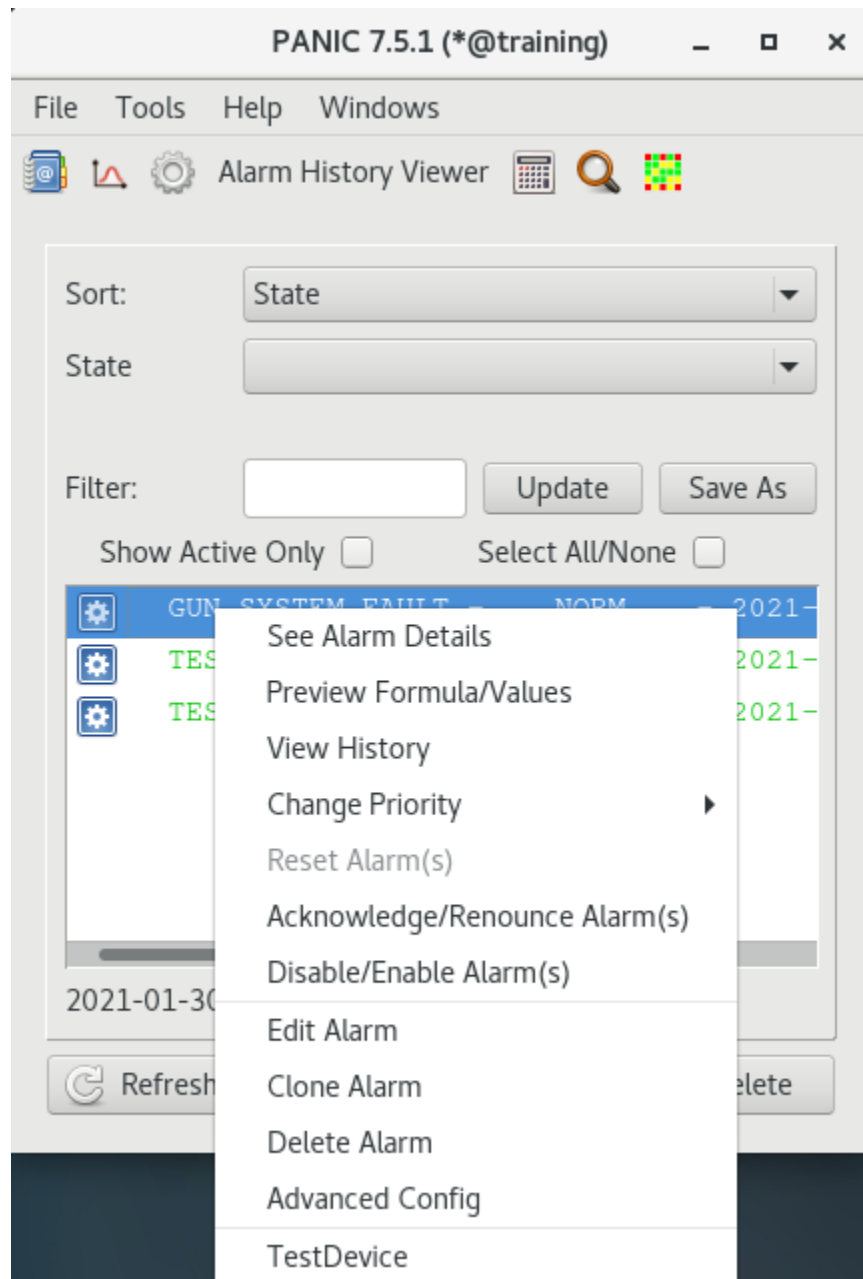


Fig. 9.39: Alarm details

9.5.1 Alarm Details

ALARM: GUN_SYSTEM_FAULT

Tag:

GUN_SYSTEM_FAULT

State:

NORM

Last Report

Reset

Disabled:

☐

Acknowledged:

☐

Device:

panic/examples/pyalarm_standard

Priority:

ALARM

Description:

Gun and HV suply state mismatch

Annunciators:

+

Formula:

((elin/gun/hv != ON)
and (elin/gun/hv/HighVoltage > 75))
and (elin/beam/run == ON)

Evaluate

Edit

Show Values



Save

Cancel


Fig. 9.40: PANIC GUI, view alarm details


9.5.2 Preview Formula/Values

GUN_SYSTEM_FAULT Alarm Formula Preview ×

Formula: ☐ Edit  

```
((elin/gun/hv !=ON)
and (elin/gun/hv/HighVoltage > 75))
and (elin/beam/run == ON)
```

 Evaluate

Values of attributes used in the Alarm formula: 

High Voltage	<div><div></div><div>75.00</div></div>	<input type="text" value="75.00"/>
--------------	--	------------------------------------

Fig. 9.41: Preview Formula/Values

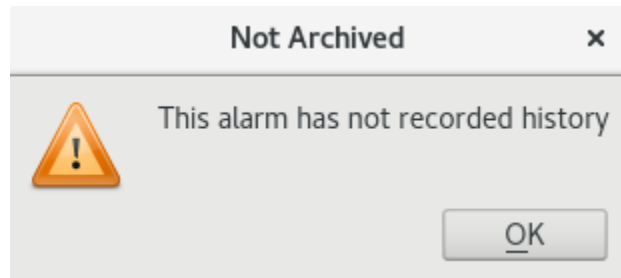


Fig. 9.42: No archivization

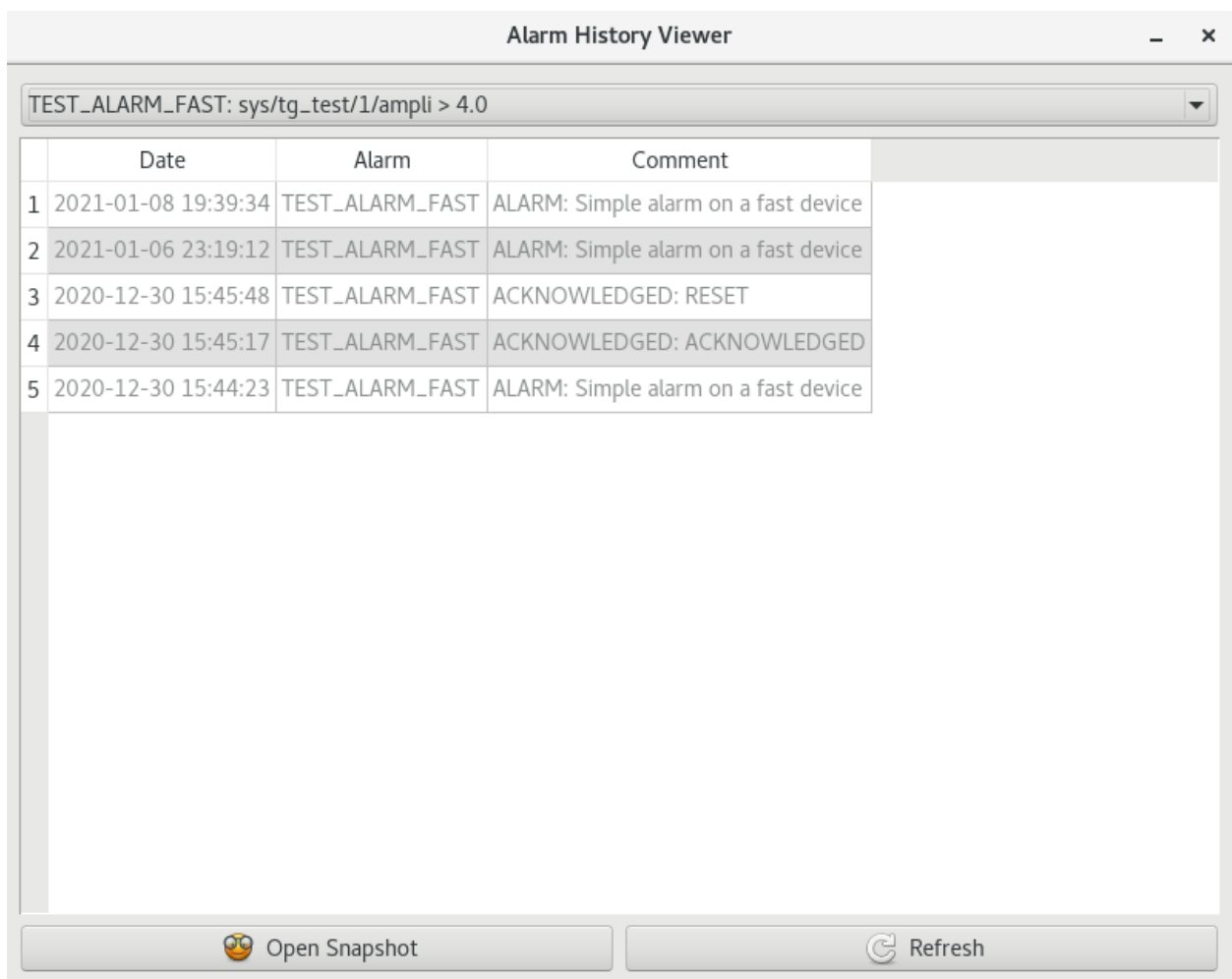


Fig. 9.43: Alarm History

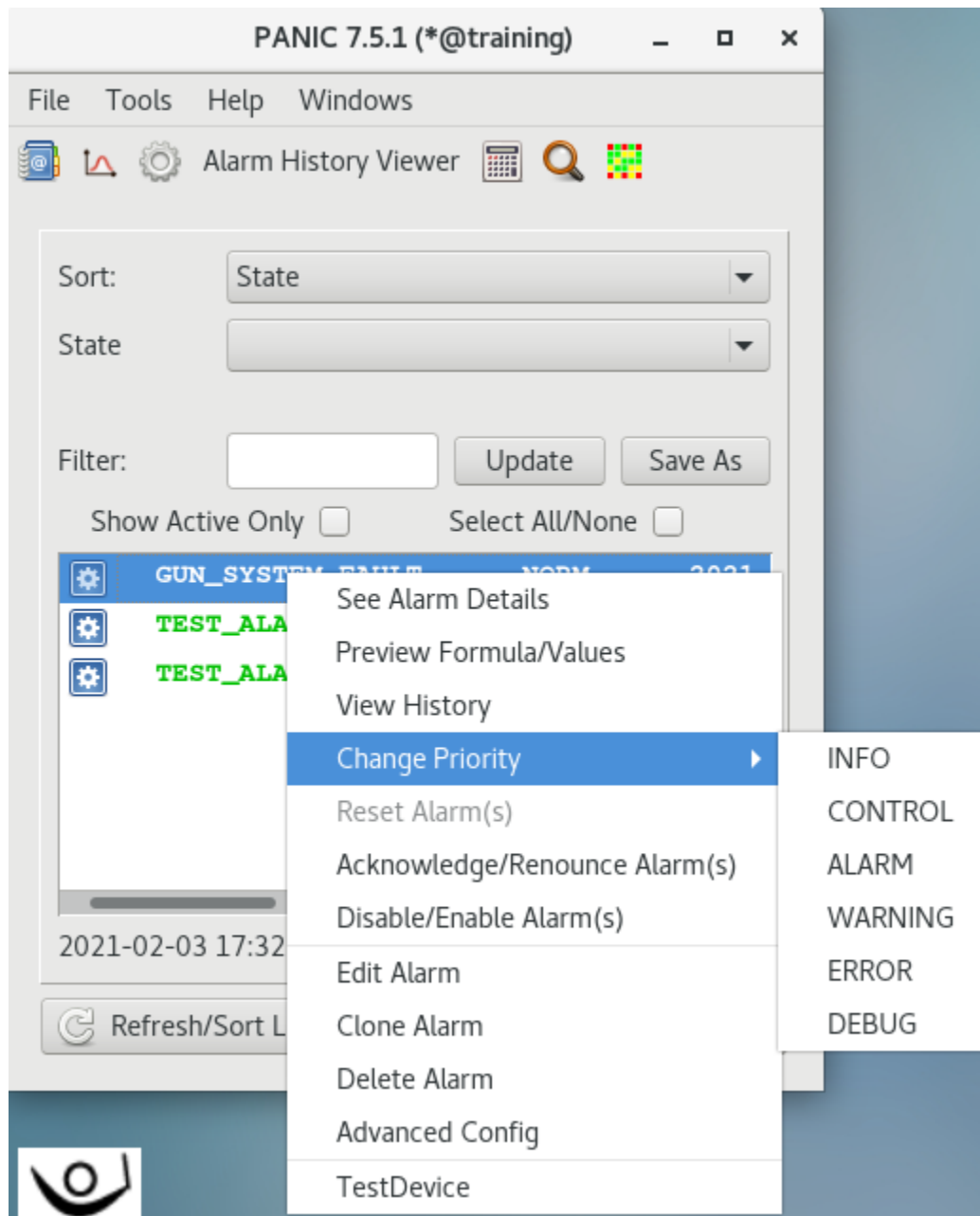


Fig. 9.44: Change Priority

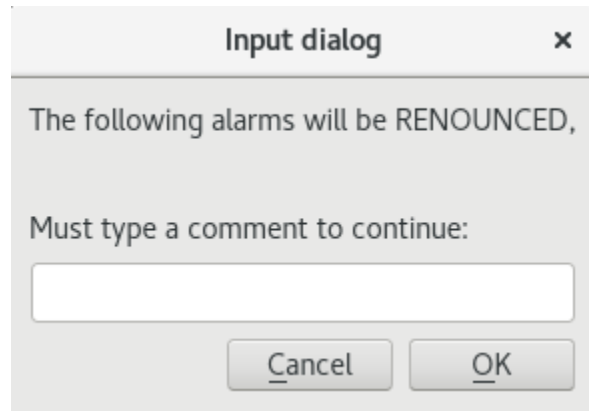


Fig. 9.45: Renounce Alarm

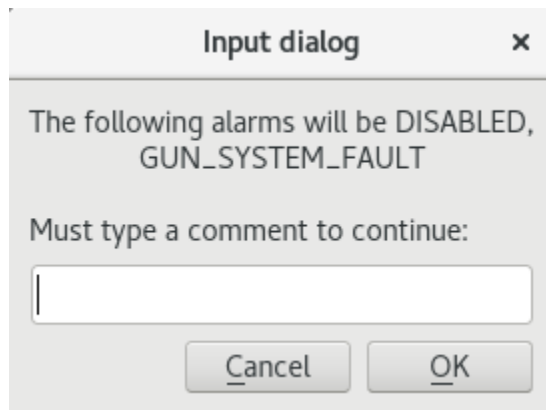


Fig. 9.46: Disable Alarm

ALARM: GUN_SYSTEM_FAULT [X]

Tag:

State: NORM

Disabled: ☐

Acknowledged: ☐

Device:

Priority:

Description:

Annunciators:

Formula:

```
((elin/gun/hv != ON)
and (elin/gun/hv/HighVoltage > 75))
and (elin/beam/run == ON)
```

Fig. 9.47: Edit Alarm

Input dialog [X]

Please provide tag name for cloned alarm.

Fig. 9.48: Clone Alarm

9.5.3 View History

9.5.4 Change Priority

9.5.5 Acknowledge/Renounce Alarm

9.5.6 Disable/Enable Alarm

9.5.7 Edit Alarm

9.5.8 Clone Alarm

9.5.9 Delete Alarm

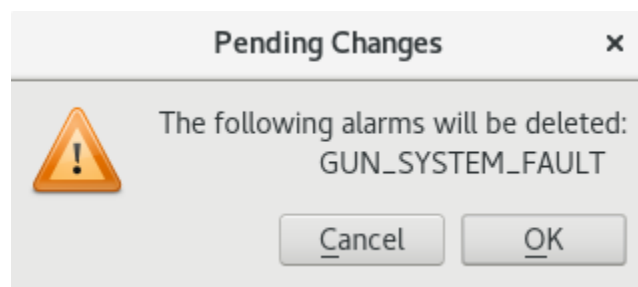


Fig. 9.49: Delete Alarm

9.5.10 Advanced Config

9.5.11 TestDevice

9.6 Alarms management

Using buttons at the bottom a user can manage alarms.

9.6.1 Create new alarm

After clicking *New* button user can configure new alarm.

9.7 Top bar menu

9.7.1 Import and Export from CSV file

Application allows to import or export to CSV file configuration of alarms

Format CSV file of CSV file must contain columns:

- TAG
- DEVICE DESCRIPTION

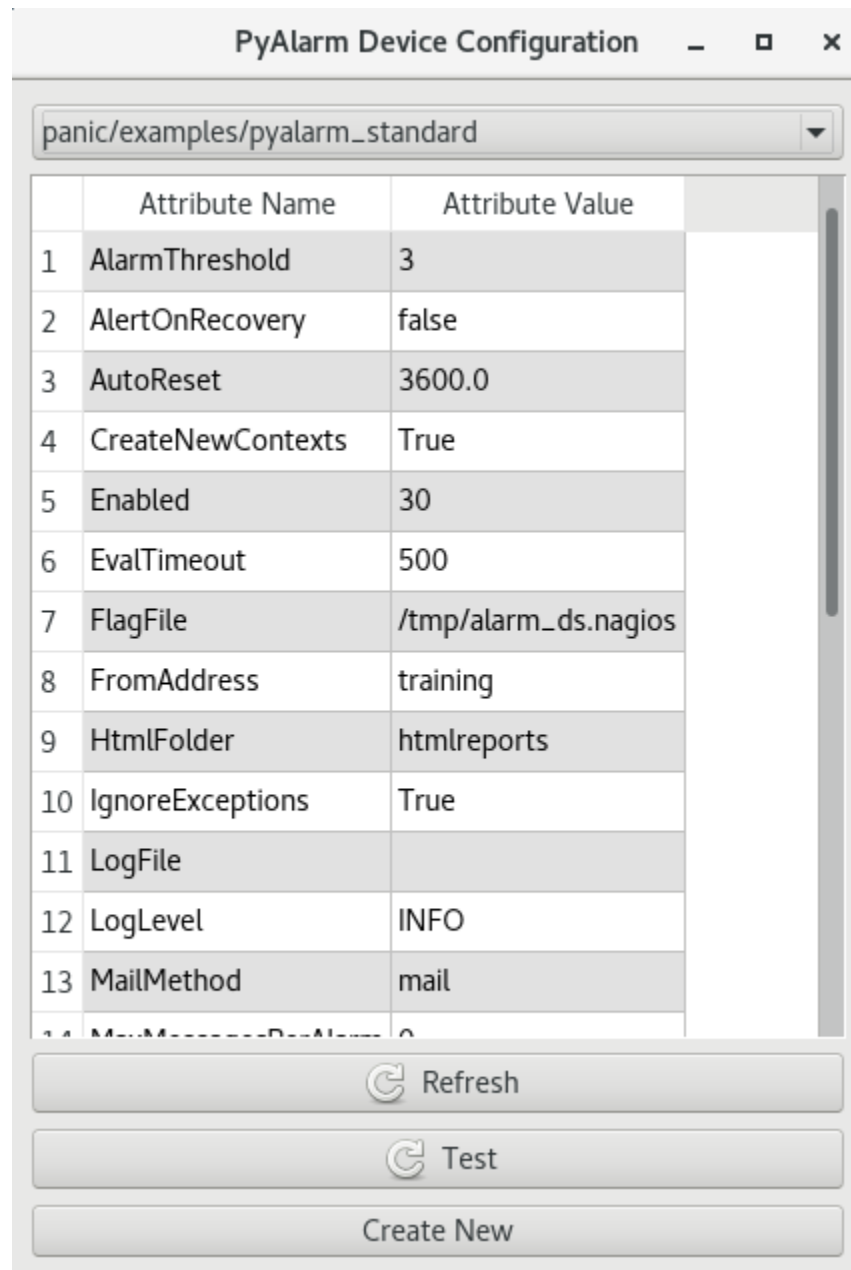


Fig. 9.50: Advanced Config

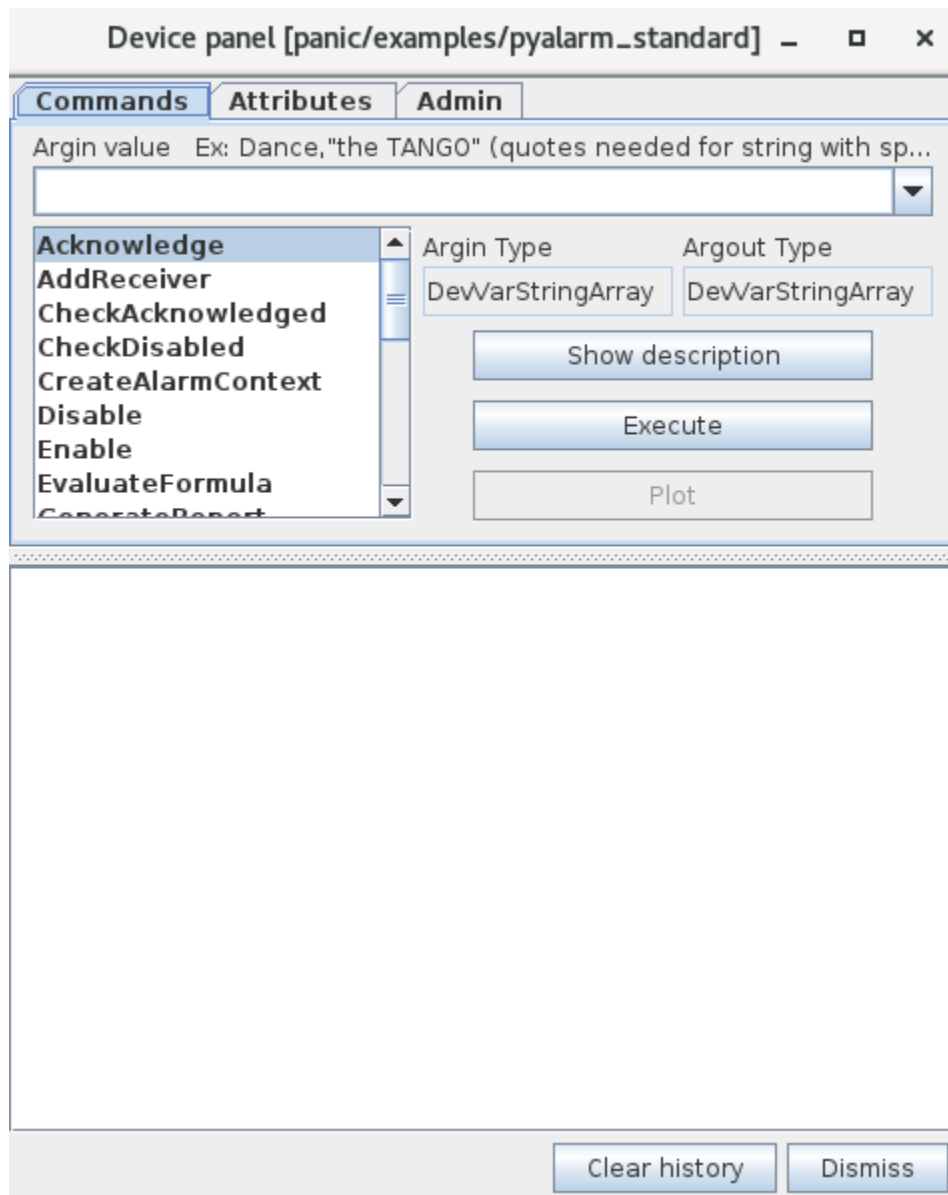


Fig. 9.51: Test Device

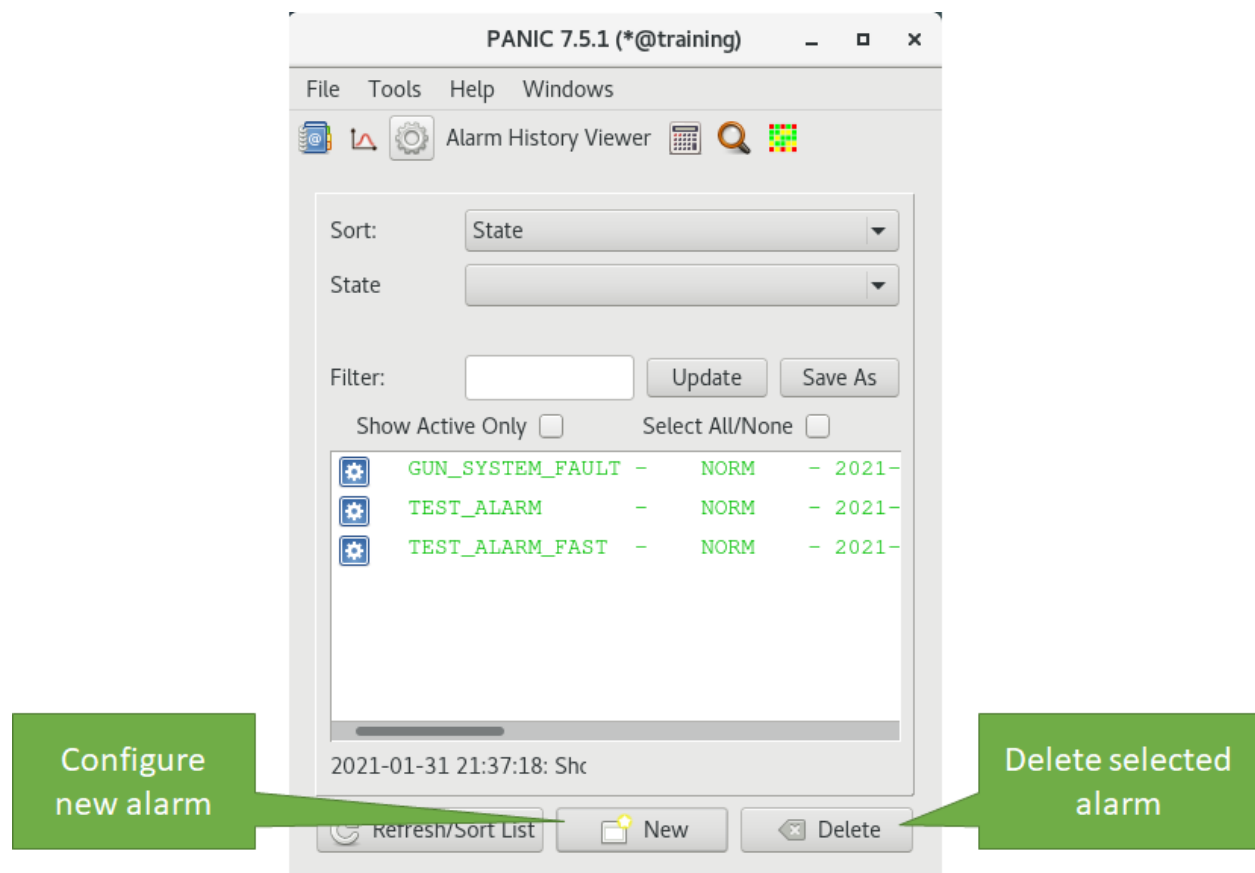


Fig. 9.52: Alarm details

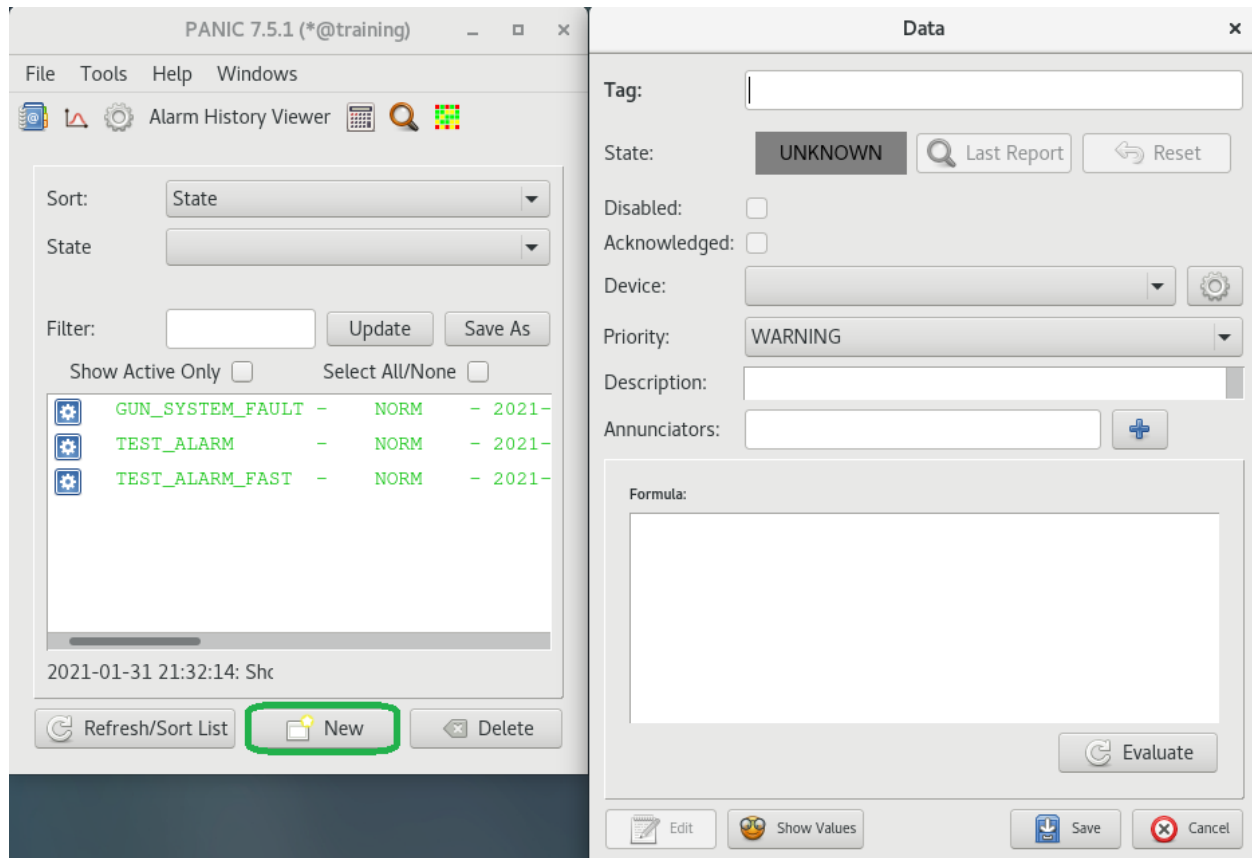


Fig. 9.53: New alarm

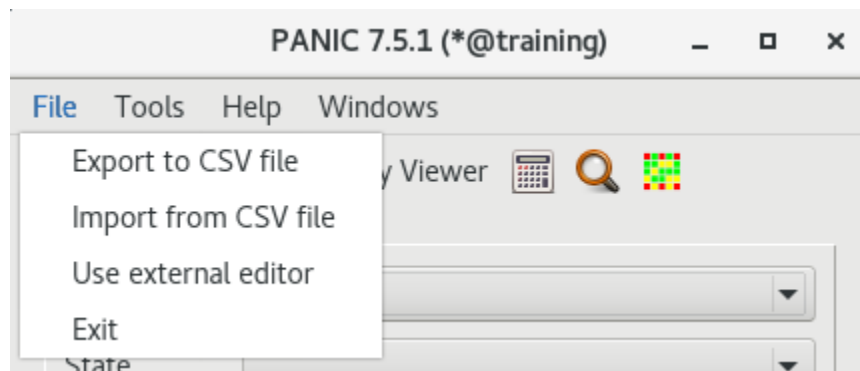
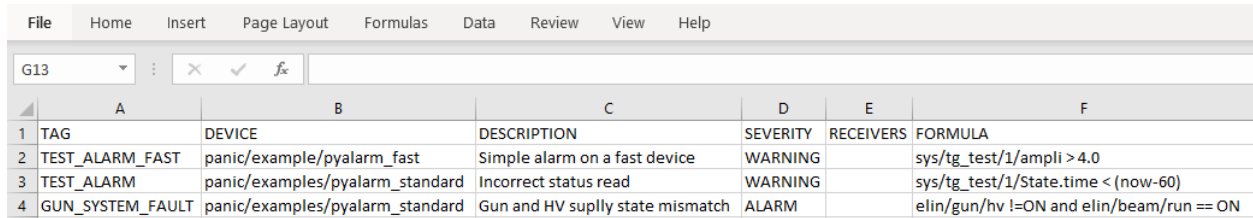


Fig. 9.54: PANIC GUI, File menu

- SEVERITY
- RECEIVERS
- FORMULA

The separation between CSV fields shall be a tab character (t). This is due to an alarm configuration containing semi-colons (;) and commas (,) which would otherwise interfere with CSV structure.



	A	B	C	D	E	F
1	TAG	DEVICE	DESCRIPTION	SEVERITY	RECEIVERS	FORMULA
2	TEST_ALARM_FAST	panic/example/pyalarm_fast	Simple alarm on a fast device	WARNING		sys/tg_test/1/ampli > 4.0
3	TEST_ALARM	panic/examples/pyalarm_standard	Incorrect status read	WARNING		sys/tg_test/1/State.time < (now-60)
4	GUN_SYSTEM_FAULT	panic/examples/pyalarm_standard	Gun and HV supply state mismatch	ALARM		elin/gun/hv != ON and elin/beam/run == ON

Fig. 9.55: CSV file example

9.7.2 Tools

PANIC GUI has included many tools to help with configuration of new alarm or edit created.

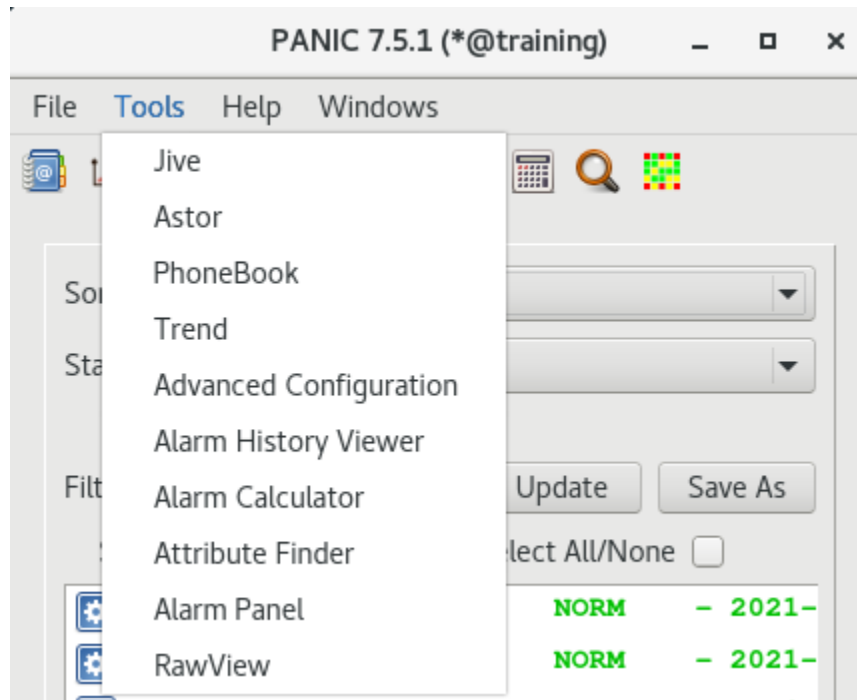


Fig. 9.56: PANIC GUI, Tools menu

ALARM PHILOSOPHY

10.1 Alarm philosophy

The alarm philosophy shall provide criteria, definitions, principles, and responsibilities for all of the alarm management lifecycle stages. It facilitates:

- consistency across the alarm system,
- consistency with risk management goals and objectives,
- good engineering practices,
- alarm system that supports an effective operator response.

10.1.1 Alarm philosophy contents

The IEC62682 standard asks for the following specification in an alarm philosophy:

- Purpose of alarm system,
- Definitions, References,
- Roles and responsibilities
- Principles of alarm design,
- Guidance for rationalisation of alarms list,
- Alarm class definitions (types of alarms, priorities, handling principles for a class),
- Alarm history preservation,
- Highly managed alarms handling (the alarms which require special procedures, i.e. due to law requirements),
- HMI/GUI design principles (colouring, symbols, naming conventions),
- Prioritisation method,
- Performance monitoring,
- Maintenance guidance,
- Testing requirements,
- Alarm documentation requirements,
- Implementation guidance,
- Management of change,
- Training,

- Related site procedures,
- Site-specific requirements,
- Audit (frequency, topics),

10.2 Alarm system requirements specification (ASRS)

The ASRS provides functional requirements specification. It is the document developed as part of the alarm philosophy stage and implementation. Its purpose is to detail the functional requirements expected of the control system. On all development stages, the ASRS shall be consistent with the alarm philosophy.

The ASRS contains a specification for some of the following:

- alarm attributes,
- alarm HMI (i.e. PANIC GUI),
- alarm communication protocol (i.e. Tango),
- alarm record logging (i.e. SNAP, elasticsearch, text files),
- alarm record analysis,
- alarm priorities available,
- visible annunciation functionality (colours, symbols),
- audible alarm annunciation functionality,
- alarm summary display functionality,
- alarm shelving, suppression,
- alarm configuration functionality
- alarm log capabilities,
- alarm monitoring and assessment functionality,
- alarm auditing functionality,
- advanced alarming functionality.

The ASRS shall be compared with capabilities if the control system / Alarm Management System tool selected. If specific criteria are not met, either ASRS shall be updated (providing that it is still compatible with the alarm philosophy) or the control system / the alarm system tools shall be corrected.

Each alarm system requirements from ASRS should be tested before the operation stage.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`