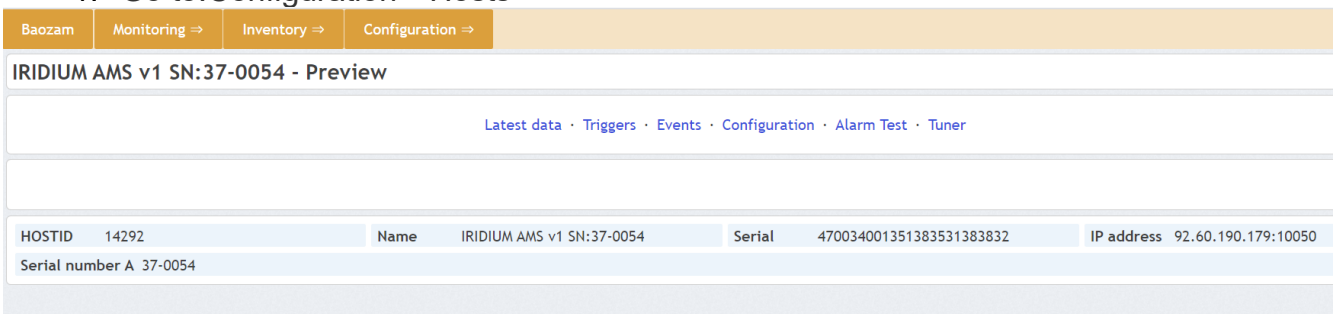
APPLICATION NOTE
AN007

How to set triggers and action in BAOZAM.
Not so quick user guide.

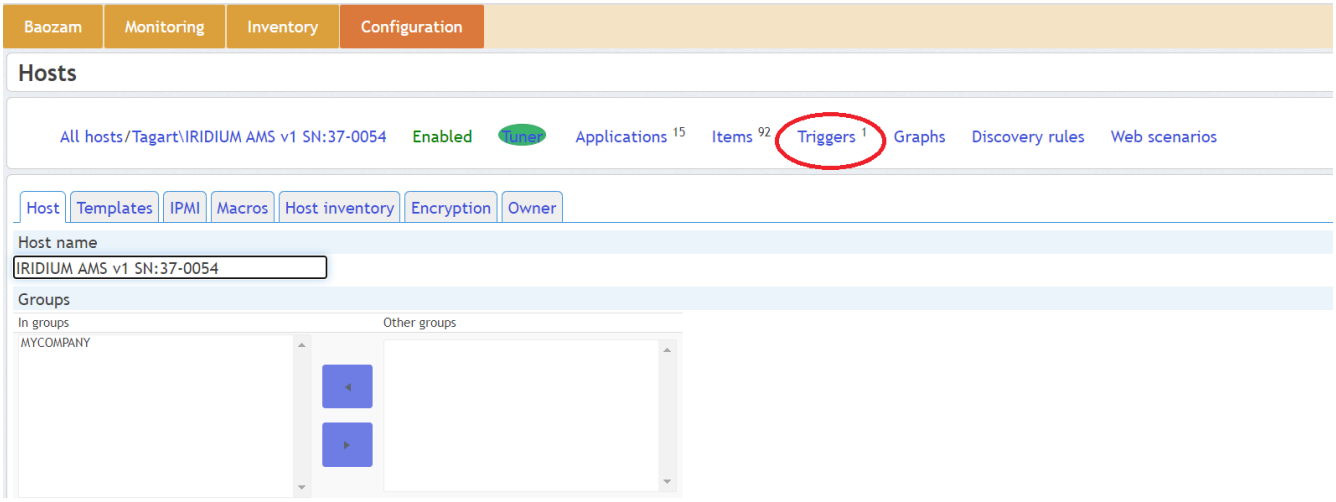
PROBLEM:
I WANT TO RECEIVE E-MAIL WHEN MY SYSTEM MAKES MORE THAN 10 ALARMS WITHIN LAST 15 MINUTES.

SOLUTION:
Configure necessary triggers and complete trigger results with necessary actions.
See below.

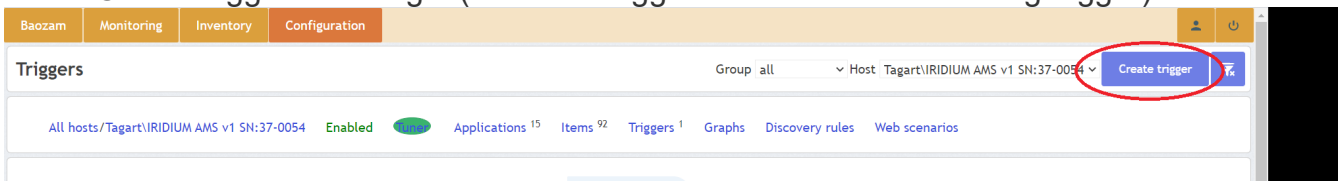
1. Go to:Configuration→Hosts



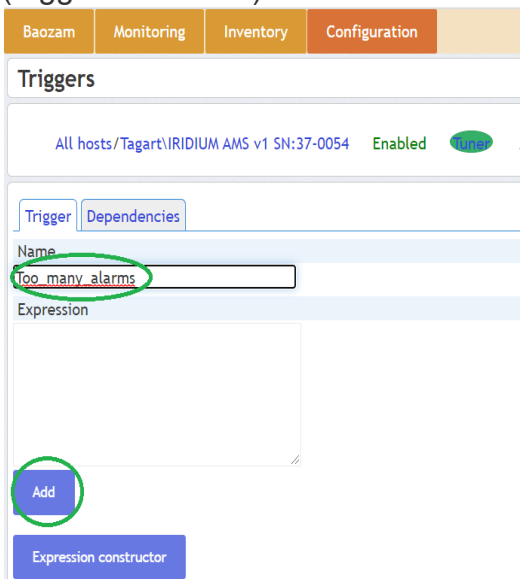
2. Click on Triggers in the row of the host



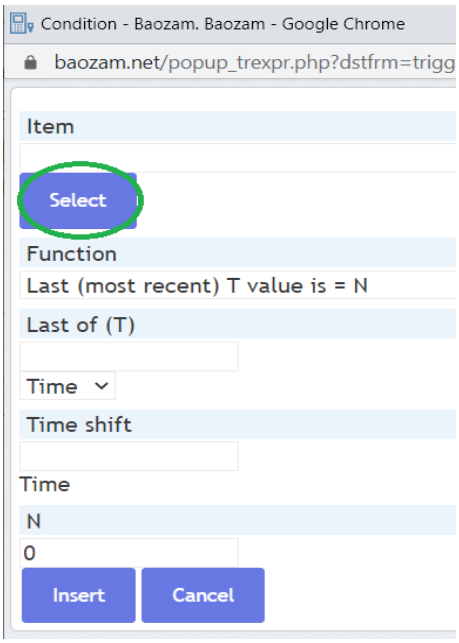
3. Click on Create trigger to the right (or on the trigger name to edit an existing trigger)



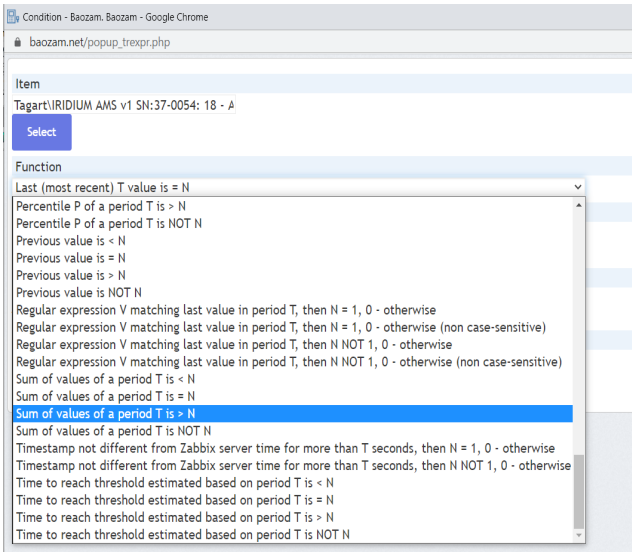
4. Enter parameters of the trigger (trigger name etc.) in the form and add/edit the conditional expression

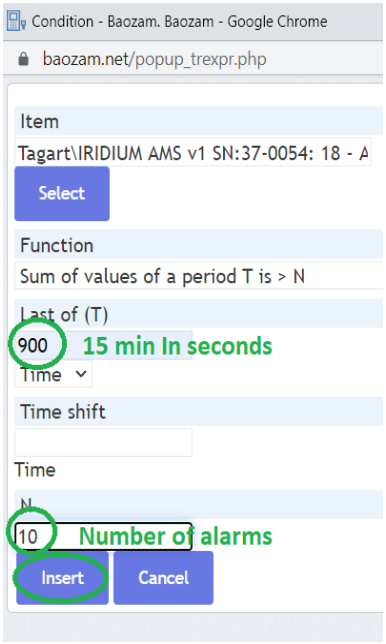


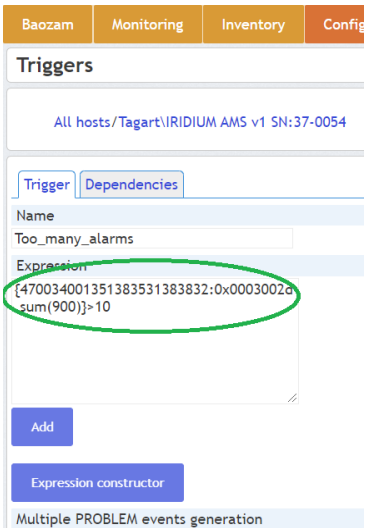
4.1. Select the item/value for the expression (for example, Alarm counter)



4.2. Select the necessary function (in our example the total number of alarms must be more than 10 during last 15 min) and fill additional parameters

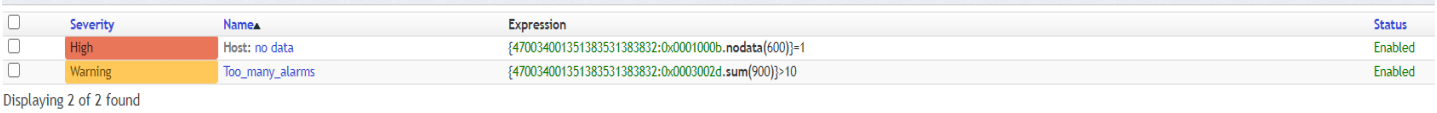






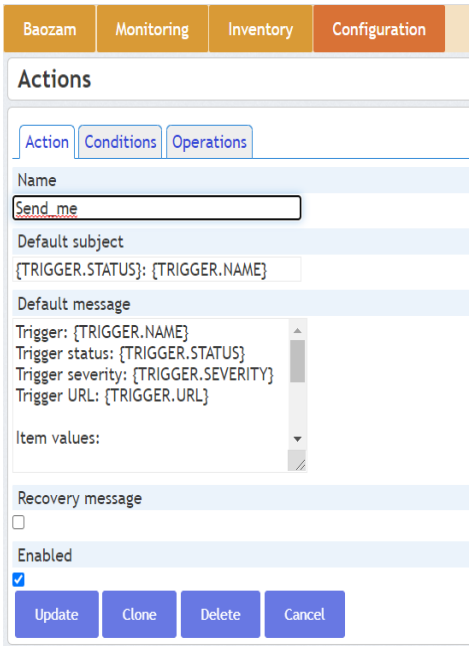
4.3. Complete the form with trigger description and severity and update this information

5. If the trigger was created successfully it appears in the trigger list. Also you can see it in your dashboard.



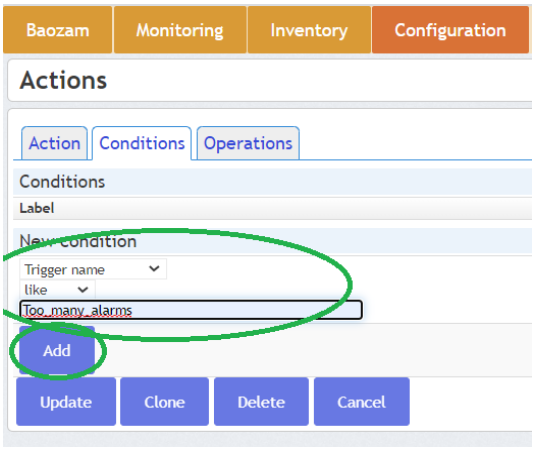
6. Create the ACTION in Configuration → Actions menu. In our example the source of event must be “Triggers”

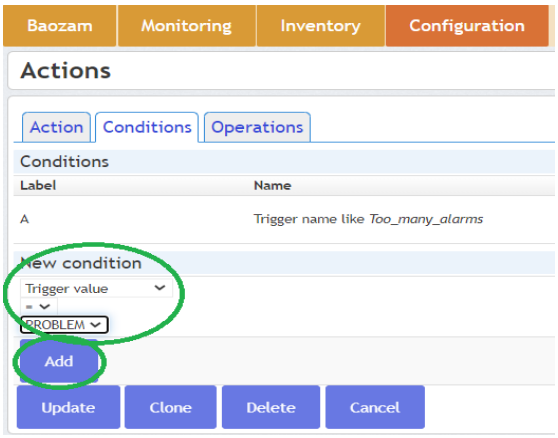
6.1. Fill the form with actions name and the message for this action

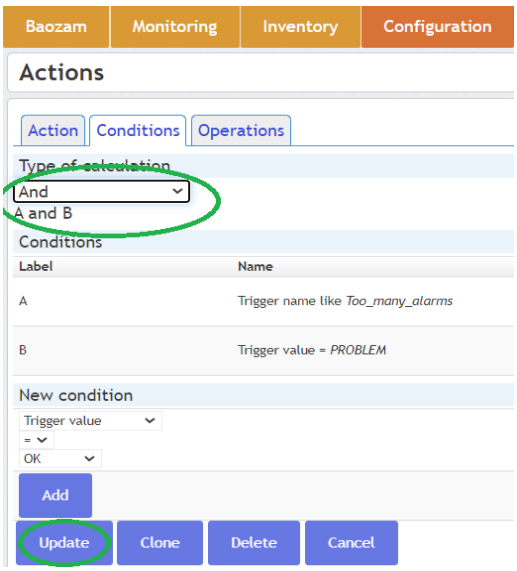


6.2. Choose the “Condition” tab and fill the logical expression. In our example the expression is something like “When the trigger called Too_many_alarms have a PROBLEM status → do something”. This expression is equal to:

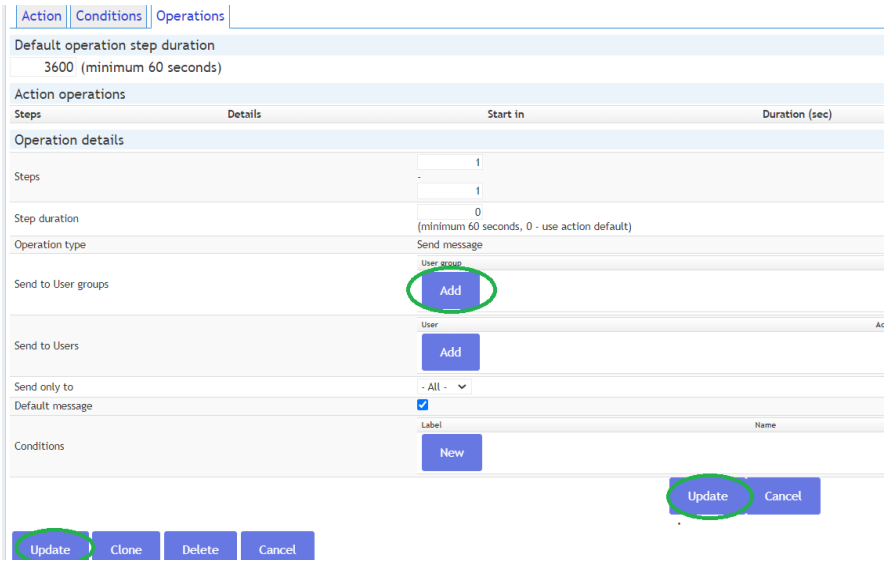
```
IF Trigger.name looks LIKE “Too_many_alarms”
AND
Thigger.value = Problem
THEN
do something
```

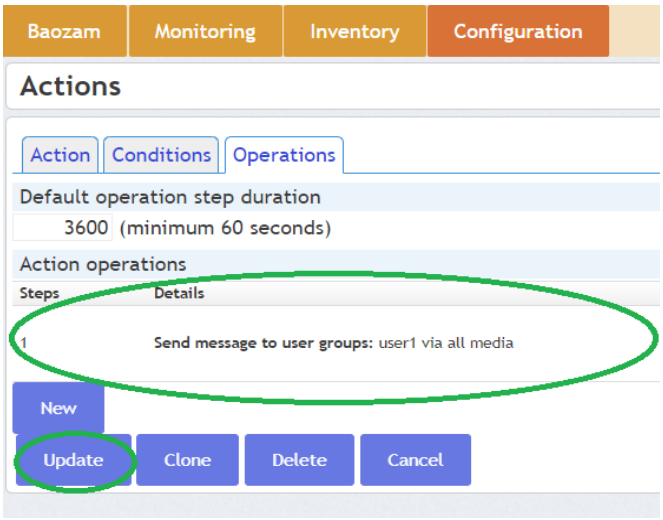




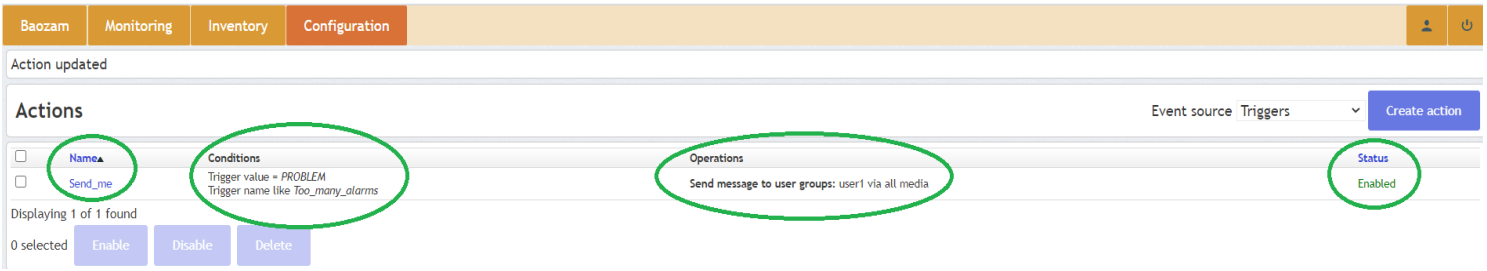


6.3. Configure the Operations tab and check the operation scenario

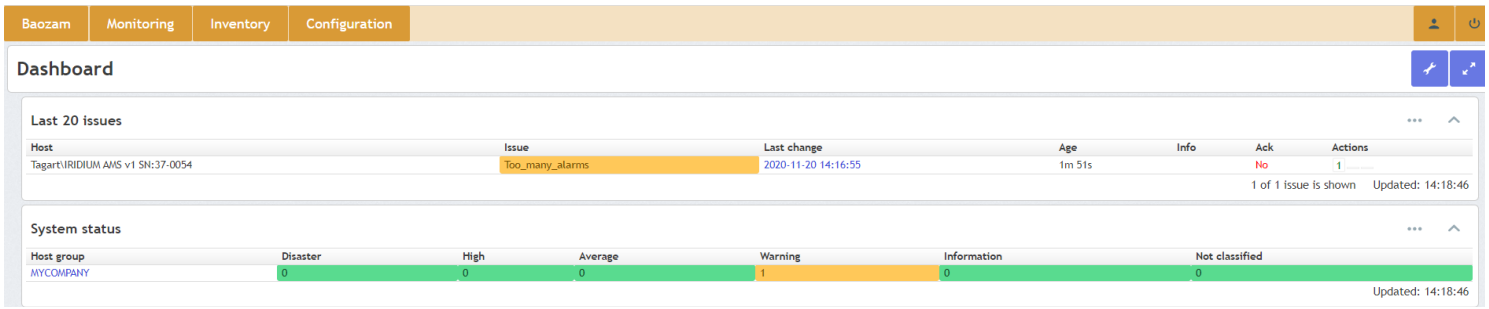




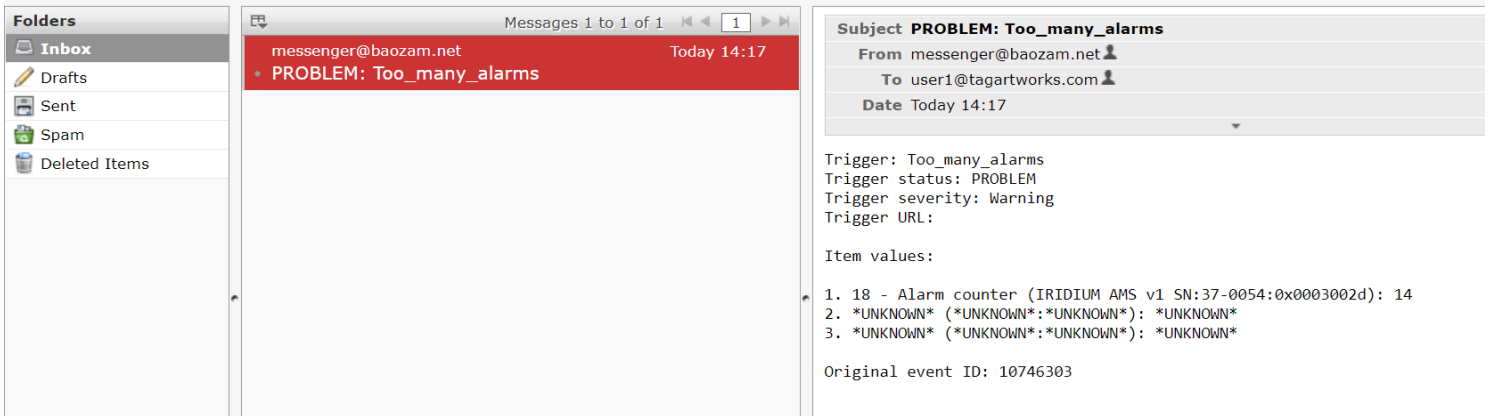
6.4. Finally check the complete action



7. TEST how it works.
   In my case I made ~20 “false positives”. In a couple of minute I received notification in my dashboard:



and in my Mailbox



VOILA

Appendix A.

Fields description for the trigger configuration

Parameter	Description
Name	Trigger name. The name may contain the supported macros: {HOST.HOST}, {HOST.NAME}, {HOST.CONN}, {HOST.DNS}, {HOST.IP}, {ITEM.VALUE}, {ITEM.LASTVALUE} and {$MACRO}. **$1, $2…$9** macros can be used to refer to the first, second…ninth constant of the expression. *Note*: $1-$9 macros will resolve correctly if referring to constants in relatively simple, straightforward expressions. For example, the name “Processor load above $1 on {HOST.NAME}” will automatically change to “Processor load above 5 on New host” if the expression is {New host:system.cpu.load[percpu,avg1].last()}>5
Severity	Set the required trigger severity by clicking the buttons.
Expression	Logical expression used to define the conditions of a problem. A problem is created after all the conditions included in the expression are met, i.e. the expression evaluates to TRUE. The problem will be resolved as soon as the expression evaluates to FALSE, unless additional recovery conditions are specified in *Recovery expression*.
PROBLEM event generation mode	Mode for generating problem events: **Single** - a single event is generated when a trigger goes into the 'Problem' state for the first time; **Multiple** - an event is generated upon *every* 'Problem' evaluation of the trigger.
URL	If not empty, the URL entered here is available as a link in several frontend locations, e.g. when clicking on the problem name in *Monitoring → Problems* (*URL* option in the *Trigger* menu) and *Problems* dashboard widget. Supported macros: {ITEM.VALUE}, {ITEM.LASTVALUE}, {TRIGGER.ID}, several {HOST.*} macros, user macros.
Description	Text field used to provide more information about this trigger. May contain instructions for fixing specific problem, contact detail of responsible staff, etc. the description may contain the same set of macros as trigger name.
Enabled	Unchecking this box will disable the trigger if required.

APPENDIX B.

TRIGGER EXPRESSIONS

The expressions used in triggers are very flexible. You can use them to create complex logical tests regarding monitored statistics.

A simple useful expression might look like:

```
{<server>:<key>.<function>(<parameter>)}<operator><constant>
```

While the syntax is exactly the same, from the functional point of view there are two types of trigger expressions:

- problem expression - defines the conditions of the problem
- recovery expression (optional) - defines additional conditions of the problem resolution

When defining a problem expression alone, this expression will be used both as the problem threshold and the problem recovery threshold. As soon as the problem expression evaluates to TRUE, there is a problem. As soon as the problem expression evaluates to FALSE, the problem is resolved.

When defining both problem expression and the supplemental recovery expression, problem resolution becomes more complex: not only the problem expression has to be FALSE, but also the recovery expression has to be TRUE. This is useful to avoid trigger flapping in hysteresis.

FUNCTIONS

Trigger functions allow to reference the collected values, current time and other factors.

FUNCTION		
Description	**Parameters**	**Comments**
abschange		
The amount of absolute difference between last and previous values.		Supported value types: float, int, str, text, log For example: (previous value;last value=abschange) 1;5=4 3;1=2 0;-2.5=2.5 For strings returns: 0 - values are equal 1 - values differ
avg (sec|#num,<time_shift>)		
Average value of an item within the defined evaluation period.	**Sec** or **#num** - maximum evaluation period in seconds or in latest collected values (preceded by a hash mark) **time_shift** (optional) - evaluation point is moved the number of seconds back in time	Supported value types: float, int Examples: ⇒avg(#5) → average value for the five latest values ⇒avg(1h) → average value for an hour ⇒avg(1h,1d) → average value for an hour one day ago. The time_shift parameter is supported. It is

FUNCTION		
Description	**Parameters**	**Comments**
abschange		
		useful when there is a need to compare the current average value with the average value time_shift seconds back.
Band (<sec|#num>,mask,<time_shift>)		
Value of “bitwise AND” of an item value and mask.	**Sec** (ignored, equals #1) or **#num** (optional) - the Nth most recent value **mask** (mandatory) - 64-bit unsigned integer (0 - 18446744073709551615) **time_shift** (optional) - see avg()	Supported value types: int Take note that #num works differently here than with many other functions (see last()). Although the comparison is done in a bitwise manner, all the values must be supplied and are returned in decimal. For example, checking for the 3rd bit is done by comparing to 4, not 100. Examples: ⇒band(,12)=8 or band(,12)=4 → 3rd or 4th bit set, but not both at the same time ⇒band(,20)=16 → 3rd bit not set and 5th bit set.
change		
The amount of difference between last and previous values.		Supported value types: float, int, str, text, log For example: (previous value;last value=change) 1;5=+4 3;1=-2 0;-2.5=-2.5 See also: abschange for comparison For strings returns: 0 - values are equal 1 - values differ
Count (sec|#num,<pattern>,<operator>,<time_shift>)		
Number of values within the defined evaluation period.	**Sec** or **#num** - maximum evaluation period in seconds or in latest collected values (preceded by a hash mark) **pattern** (optional) - required pattern **operator** (optional) Supported operators:	Supported value types: float, integer, string, text, log Float items match with the precision of 0.000001. With *band* as third parameter, the second pattern parameter can be specified as two numbers, separated by '/': **number_to_compare_with/mask**. count() calculates “bitwise AND” from the value and

FUNCTION		
Description	**Parameters**	**Comments**
abschange		
	eq - equal *ne* - not equal *gt* - greater *ge* - greater or equal *lt* - less *le* - less or equal *like* - matches if contains pattern (case-sensitive) *band* - bitwise AND *regexp* - case sensitive match of regular expression given in pattern *iregexp* - case insensitive match of regular expression given in pattern Note that: *eq* (default), *ne*, *gt*, *ge*, *lt*, *le*, *band*, *regexp*, *regexp* are supported for integer items *eq* (default) *ne*, *gt*, *ge*, *lt*, *le*, *regexp* *iregexp* are supported for float items *like* (default), *eq*, *ne*, *regexp*, *iregexp* are supported for string, text and log items **time_shift** (optional) - see avg()	the *mask* and compares the result to *number_to_compare_with*. If the result of “bitwise AND” is equal to *number_to_compare_with*, the value is counted. If *number_to_compare_with* and *mask* are equal, only the *mask* need be specified (without '/'). With *regexp* or *iregexp* as third parameter the second pattern parameter can be an ordinary or global (starting with '@') regular expression. In case of global regular expressions case sensitivity is inherited from global regular expression settings. For the purpose of regexp matching, float values will always be represented with 4 decimal digits after '.'. Also note that for large numbers difference in decimal (stored in database) and binary representation may affect the 4th decimal digit. Examples: ⇒count(10m) → number of values for last 10 minutes ⇒count(10m,"error",eq) → number of values for last 10 minutes that equal 'error' ⇒count(10m,12) → number of values for last 10 minutes that equal '12' ⇒count(10m,12,gt) → number of values for last 10 minutes that are over '12' ⇒count(#10,12,gt) → number of values within last 10 values that are over '12' ⇒count(10m,12,gt,1d) → number of values for preceding 10 minutes up to 24 hours ago that were over '12' ⇒count(10m,6/7,band) → number of values for last 10 minutes having '110' (in binary) in the 3 least significant bits. ⇒count(10m,,,1d) → number of values for preceding 10 minutes up to 24 hours ago
date		
Current date in YYYYMMDD format.		Supported value types: *any* Example of returned value: 20150731
dayofmonth		

FUNCTION		
Description	**Parameters**	**Comments**
abschange		
Day of month in range of 1 to 31.		Supported value types: *any*
dayofweek		
Day of week in range of 1 to 7 (Mon - 1, Sun - 7).		Supported value types: *any*
delta (sec|#num,<time_shift>)		
Difference between the maximum and minimum values within the defined evaluation period ('max()' minus 'min()').	**Sec** or **#num** - maximum evaluation period in seconds or in latest collected values specified (preceded by a hash mark) **time_shift** (optional) - see avg()	Supported value types: float, int
diff		
Checking if last and previous values differ.		Supported value types: float, int, str, text, log Returns: 1 - last and previous values differ 0 - otherwise
forecast (sec|#num,<time_shift>,time,<fit>,<mode>)		
Future value, max, min, delta or avg of the item.	**Sec** or **#num** maximum evaluation period in seconds or in latest collected values specified (preceded by a hash mark) **time_shift** (optional) - see avg() **time** - forecasting horizon in seconds **fit** (optional) - function used to fit historical data Supported fits: *linear* - linear function *polynomialN* - polynomial of degree N (1 <= N <= 6) *exponential* - exponential function *logarithmic* - logarithmic function	Supported value types: float, int If value to return is larger than 999999999999.9999 or less than -999999999999.9999, return value is cropped to 999999999999.9999 or -999999999999.9999 correspondingly. Becomes not supported only if misused in expression (wrong item type, invalid parameters), otherwise returns -1 in case of errors. Examples: ⇒forecast(#10,,1h) → forecast of item value after one hour based on last 10 values

FUNCTION		
Description	**Parameters**	**Comments**
abschange		
	power - power function Note that: *linear* is default, *polynomial1* is equivalent to *linear*	⇒forecast(1h,,30m) → forecast of item value after 30 minutes based on last hour data ⇒forecast(1h,1d,12h) → forecast of item after 12 hours based on one hour one day ago ⇒forecast(1h,,10m,exponential) → forecast of item value after 10 minutes based on last hour data and exponential function ⇒forecast(1h,,2h,polynomial3,max) → forecast of maximum value item can reach in next two hours based on last hour data and cubic (third degree) polynomial ⇒forecast(# 2, -20m) → estimate the value of an item which was 20 minutes ago based on last two values (this can be more precise than using last() or prev(), especially if item is updated rarely, say, once an hour)
	mode (optional) - demanded output Supported modes: *value* - value (default) *max* - maximum *min* - minimum *delta* - *max-min* *avg* - average Note that: *value* estimates item value at the moment now + time *max*, *min*, *delta* and *avg* investigate item value estimate on the interval between now and now + time	
Fuzzytime (sec)		
Checking how much an item value (as timestamp) differs from the server time.	**Sec** - seconds	Supported value types: float, int Returns: 1 - difference between item value (as timestamp) and server timestamp is less than or equal to T seconds 0 - otherwise Example: ⇒fuzzytime(60)=0 → detect a problem if time difference is over 60 seconds
Iregexp (<pattern>,<sec|#num>)		
This function is a non case-sensitive analogue of regexp().	see regexp()	Supported value types: str, log, text
last(<sec|#num>,<time_shift>)		

FUNCTION		
Description	**Parameters**	**Comments**
abschange		
The most recent value.	**Sec** (ignored, equals #1) or **#num** (optional) - the Nth most recent value **time_shift** (optional) - see avg()	Supported value types: float, int, str, text, log Take note that #num works differently here than with many other functions. For example: last() is always equal to last(#1) last(#3) - third most recent value (*not* three latest values) Server does not guarantee exact order of values if more than two values exist within one second in history.
Max (sec|#num,<time_shift>)		
Highest value of an item within the defined evaluation period.	**Sec** or **#num** maximum evaluation period in seconds or in latest collected values (preceded by a hash mark) **time_shift** (optional) - see avg()	Supported value types: float, int
Min (sec|#num,<time_shift>)		
Lowest value of an item within the defined evaluation period.	**Sec** or **#num** - maximum evaluation period in seconds or in latest collected values (preceded by a hash mark) **time_shift** (optional) - see avg()	Supported value types: float, int
nodata (sec)		
Checking for no data received.	**Sec** - evaluation period in seconds. The period should not be less than 30 seconds because the history syncer process calculates this function only every 30 seconds. nodata(0) is disallowed.	Supported value types: *any* Returns: 1 - if no data received during the defined period of time 0 - otherwise Note that this function will display an error if, within the period of the 1st parameter: - there's no data and server was restarted - there's no data and maintenance was completed - there's no data and the item was added or re-enabled
now		

FUNCTION		
Description	**Parameters**	**Comments**
abschange		
Number of seconds since the Epoch (00:00:00 UTC, January 1, 1970).		Supported value types: *any*
prev		
Previous value.		Supported value types: float, int, str, text, log Returns the same as last(#2).
Str (<pattern>,<sec|#num>)		
Finding a string in the latest (most recent) value.	**Pattern** (optional) - required string **sec** or #**num** (optional) - maximum evaluation period in seconds or in latest collected values (preceded by a hash mark). In this case, more than one value may be processed.	Supported value types: str, text, log Returns: 1 - found 0 - otherwise If more than one value is processed, '1' is returned if there is at least one matching value. This function is case-sensitive.
Strlen (<sec|#num>,<time_shift>)		
Length of the latest (most recent) value in characters (not bytes).	**Sec** (ignored, equals #1) or **#num** (optional) - the Nth most recent value **time_shift** (optional) - see avg()	Supported value types: str, text, log Take note that #num works differently here than with many other functions. Examples: ⇒strlen()(is equal to strlen(#1)) → length of the latest value ⇒strlen(#3) → length of the third most recent value ⇒strlen(,1d) → length of the most recent value one day ago.
Sum (sec|#num,<time_shift>)		
Sum of collected values within the defined evaluation period.	**Sec** or **#num** - maximum evaluation period in seconds or in latest collected values (preceded by a hash mark) **time_shift** (optional) - see avg()	Supported value types: float, int

FUNCTION		
Description	**Parameters**	**Comments**
abschange		
time		
Current time in HHMMSS format.		Supported value types: *any* Example of returned value: 123055
Timeleft (sec|#num,<time_shift>,threshold,<fit>)		
Time in seconds needed for an item to reach a specified threshold.	**Sec** or **#num** - maximum evaluation period in seconds or in latest collected values (preceded by a hash mark) **time_shift** (optional) - see avg() **threshold** - value to reach **fit** (optional) - see forecast()	Supported value types: float, int If value to return is larger than 999999999999.9999, return value is cropped to 999999999999.9999. Returns 999999999999.9999 if threshold cannot be reached. Becomes not supported only if misused in expression (wrong item type, invalid parameters), otherwise returns -1 in case of errors. Examples: ⇒timeleft(#10,,0) → time until item value reaches zero based on last 10 values ⇒timeleft(1h,,100) → time until item value reaches 100 based on last hour data ⇒timeleft(1h,1d,0) → time until item value reaches 0 based on one hour one day ago ⇒timeleft(1h,,200,polynomial2) → time until item reaches 200 based on last hour data and assumption that item behaves like quadratic (second degree) polynomial

FUNCTION PARAMETERS

Most of numeric functions accept the number of seconds as a parameter.
You may use the prefix”#”to specify that a parameter has a different meaning:

FUNCTION CALL	MEANING
sum(600)	Sum of all values in no more than the latest 600 seconds
sum(#5)	Sum of all values in no more than the last 5 values

The function”last”uses a different meaning for values when prefixed with the hash mark - it makes it choose the n-th previous value, so given the values 3, 7, 2, 6, 5 (from most recent to least recent),”last(#2)”would return 7 and”last(#5)”would return 5.

Several functions support an additional, second ”time_shift” parameter. This parameter allows to reference data from a period of time in the past. For example,”avg(1h,1d)”will return the average value for an hour one day ago.

You can use the supported ”unit symbols“ in trigger expressions, for example '5m' (minutes) instead of '300' seconds or '1d' (day) instead of '86400' seconds. '1K' will stand for '1024' bytes.
Numbers with a '+' sign are not supported.

OPERATORS

The following operators are supported for triggers (in descending priority of execution):

PRIORITY	OPERATOR	DEFINITION	Notes for”unknown values”
1	**-**	Unary minus	**-**Unknown → Unknown
2	**not**	Logical NOT	**not** Unknown → Unknown
3	*****	Multiplication	0 ***** Unknown → Unknown (yes, Unknown, not 0 - to not lose Unknown in arithmetic operations) 1.2 ***** Unknown → Unknown
	/	Division	Unknown **/** 0 → error Unknown **/** 1.2 → Unknown 0.0 **/** Unknown → Unknown
4	**+**	Arithmetical plus	1.2 **+** Unknown → Unknown
	-	Arithmetical minus	1.2 **-** Unknown → Unknown
5	**<**	Less than. The operator is defined as: A<B ⇔(A<B-0.000001)	1.2 **<** Unknown → Unknown
	<=	Less than or equal to. The operator is defined as: A<=B ⇔(A≤B+0.000001)	Unknown **<=** Unknown → Unknown
	>	More than. The operator is defined as: A>B ⇔(A>B+0.000001)	
	>=	More than or equal to. The operator is defined as: A>=B ⇔(A≥B-0.000001)	
6	**=**	Is equal. The operator is defined as: A=B ⇔(A≥B-0.000001) and (A≤B+0.000001)	
	<>	Not equal. The operator is defined as: A<>B ⇔(A<B-0.000001) or (A>B+0.000001)	

PRIORITY	OPERATOR	DEFINITION	Notes for”unknown values”
7	**and**	Logical AND	0 **and** Unknown → 0 1 **and** Unknown → Unknown Unknown **and** Unknown → Unknown
8	**or**	Logical OR	1 **or** Unknown → 1 0 **or** Unknown → Unknown Unknown **or** Unknown → Unknown

Not,”and” and “or”operators are case-sensitive and must be in lowercase. They also must be surrounded by spaces or parentheses.
All operators, except unary”-”and ”not”, have left-to-right associativity. Unary -/and/not/are non-associative (meaning -(-1) and not (not 1) should be used instead of –1 and not not 1).
Evaluation result:

- <, <=, >, >=, =, <> operators shall yield '1' in the trigger expression if the specified relation is true and '0' if it is false. If at least one operand is Unknown the result is Unknown;
- and for known operands shall yield '1' if both of its operands compare unequal to '0'; otherwise, it yields '0'; for unknown operands and yields '0' only if one operand compares equal to '0'; otherwise, it yields 'Unknown';
- or for known operands shall yield '1' if either of its operands compare unequal to '0'; otherwise, it yields '0'; for unknown operands or yields '1' only if one operand compares unequal to '0'; otherwise, it yields 'Unknown';
- The result of the logical negation operator *not* for a known operand is '0' if the value of its operand compares unequal to '0'; '1' if the value of its operand compares equal to '0'. For unknown operand *not* yields 'Unknown'.

VALUE CACHING

Values required for trigger evaluation are cached by server. Because of this trigger evaluation causes a higher database load for some time after the server restarts. The value cache is not cleared when item history values are removed (either manually or by housekeeper), so the server will use the cached values until they are older than the time periods defined in trigger functions or server is restarted.