2 Trigger expression

Overview

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.

A complete list of supported functions is available.

Typically functions return numeric values. However, returning strings for string comparison is also possible with **=** and **<>** operators; see operators and trigger examples for more details.

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 latest 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	Force cast operand to float [1]
1	**-**	Unary minus	**-**Unknown → Unknown	Yes
2	**not**	Logical NOT	**not** Unknown → Unknown	Yes
3	*	Multiplication	0 * Unknown → Unknown (yes, Unknown, not 0 - to not lose Unknown in arithmetic operations) 1.2 * Unknown → Unknown	Yes
	/	Division	Unknown / 0 → error Unknown / 1.2 → Unknown 0.0 / Unknown → Unknown	Yes
4	**+**	Arithmetical plus	1.2 **+** Unknown → Unknown	Yes
	-	Arithmetical minus	1.2 **-** Unknown → Unknown	Yes
5	**<**	Less than. The operator is defined as: A<B ⇔ (A<B-0.000001)	1.2 **<** Unknown → Unknown	Yes
	<=	Less than or equal to. The operator is defined as: A<=B ⇔ (A≤B+0.000001)	Unknown **<=** Unknown → Unknown	Yes
	>	More than. The operator is defined as: A>B ⇔ (A>B+0.000001)		Yes
	>=	More than or equal to. The operator is defined as: A>=B ⇔ (A≥B-0.000001)		Yes
6	**=**	Is equal. The operator is defined as: A=B ⇔ (A≥B-0.000001) and (A≤B+0.000001)		No [1]

Priority	Operator	Definition	Notes for unknown values	Force cast operand to float [1]
	<>	Not equal. The operator is defined as: A<>B ⇔ (A<B-0.000001) or (A>B+0.000001)		No [1]
7	**and**	Logical AND	0 **and** Unknown → 0 1 **and** Unknown → Unknown Unknown **and** Unknown → Unknown	Yes
8	**or**	Logical OR	1 **or** Unknown → 1 0 **or** Unknown → Unknown Unknown **or** Unknown → Unknown	Yes

[1] String operand is still cast to numeric if:

- another operand is numeric
- operator other than **=** or **<>** is used on an operand

(If the cast fails - numeric operand is cast to a string operand and both operands get compared as strings.)

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 Zabbix 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.

Examples of triggers

Example 1

Processor load is too high on www.zabbix.com

```
{www.zabbix.com:system.cpu.load[all,avg1].last()}>5
```

'www.zabbix.com:system.cpu.load[all,avg1]' gives a short name of the monitored parameter. It specifies that the server is 'www.zabbix.com' and the key being monitored is 'system.cpu.load[all,avg1]'. By using the function 'last()', we are referring to the most recent value. Finally, '>5' means that the trigger is in the PROBLEM state whenever the most recent processor load measurement from www.zabbix.com is greater than 5.

Example 2

www.zabbix.com is overloaded

```
{www.zabbix.com:system.cpu.load[all,avg1].last()}>5 or
{www.zabbix.com:system.cpu.load[all,avg1].min(10m)}>2
```

The expression is true when either the current processor load is more than 5 or the processor load was more than 2 during last 10 minutes.

Example 3

/etc/passwd has been changed

Use of function diff:

```
{www.zabbix.com:vfs.file.cksum[/etc/passwd].diff()}=1
```

The expression is true when the previous value of checksum of /etc/passwd differs from the most recent one.

Similar expressions could be useful to monitor changes in important files, such as /etc/passwd, /etc/inetd.conf, /kernel, etc.

Example 4

Someone is downloading a large file from the Internet

Use of function min:

```
{www.zabbix.com:net.if.in[eth0,bytes].min(5m)}>100K
```

The expression is true when number of received bytes on eth0 is more than 100 KB within last 5 minutes.

Example 5

Both nodes of clustered SMTP server are down

Note use of two different hosts in one expression:

```
{smtp1.zabbix.com:net.tcp.service[smtp].last()}=0 and
{smtp2.zabbix.com:net.tcp.service[smtp].last()}=0
```

The expression is true when both SMTP servers are down on both smtp1.zabbix.com and smtp2.zabbix.com.

Example 6

Zabbix agent needs to be upgraded

Use of function str():

```
{zabbix.zabbix.com:agent.version.str("beta8")}=1
```

The expression is true if Zabbix agent has version beta8 (presumably 1.0beta8).

Example 7

Server is unreachable

```
{zabbix.zabbix.com:icmpping.count(30m,0)}>5
```

The expression is true if host “zabbix.zabbix.com” is unreachable more than 5 times in the last 30 minutes.

Example 8

No heartbeats within last 3 minutes

Use of function nodata():

```
{zabbix.zabbix.com:tick.nodata(3m)}=1
```

To make use of this trigger, 'tick' must be defined as a Zabbix trapper item. The host should periodically send data for this item using zabbix_sender. If no data is received within 180 seconds, the trigger value becomes PROBLEM.

Note that 'nodata' can be used for any item type.

Example 9

CPU activity at night time

Use of function time():

```
{zabbix:system.cpu.load[all,avg1].min(5m)}>2 and
{zabbix:system.cpu.load[all,avg1].time()}>000000 and
{zabbix:system.cpu.load[all,avg1].time()}<060000
```

The trigger may change its status to true, only at night (00:00-06:00) time.

Example 10

Check if client local time is in sync with Zabbix server time

Use of function fuzzytime():

```
{MySQL_DB:system.localtime.fuzzytime(10)}=0
```

The trigger will change to the problem state in case when local time on server MySQL_DB and Zabbix server differs by more than 10 seconds. Note that 'system.localtime' must be configured as a passive check.

Example 11

Comparing average load today with average load of the same time yesterday (using a second `time_shift` parameter).

```
{server:system.cpu.load.avg(1h)}/{server:system.cpu.load.avg(1h,1d)}>2
```

This expression will fire if the average load of the last hour tops the average load of the same hour yesterday more than two times.

Example 12

Using the value of another item to get a trigger threshold:

```
{Template PfSense:hrStorageFree[{#SNMPVALUE}].last()}<{Template
PfSense:hrStorageSize[{#SNMPVALUE}].last()}*0.1
```

The trigger will fire if the free storage drops below 10 percent.

Example 13

Using evaluation result to get the number of triggers over a threshold:

```
({server1:system.cpu.load[all,avg1].last()}>5) +
({server2:system.cpu.load[all,avg1].last()}>5) +
({server3:system.cpu.load[all,avg1].last()}>5)>=2
```

The trigger will fire if at least two of the triggers in the expression are over 5.

Example 14

Comparing string values of two items - operands here are functions that return strings.

Problem: create an alert if Ubuntu version is different on different hosts

```
{Riga Zabbix server:vfs.file.contents[/etc/os-release].last()}<>{London
Zabbix server:vfs.file.contents[/etc/os-release].last()}
```

Example 15

Comparing two string values - operands are:

- a function that returns a string
- a combination of macros and strings

Problem: detect changes in the DNS query

The item key is:

```
net.dns.record[8.8.8.8,{$WEBSITE_NAME},{$DNS_RESOURCE_RECORD_TYPE},2,1]
```

with macros defined as

```
{$WEBSITE_NAME} = zabbix.com
{$DNS_RESOURCE_RECORD_TYPE} = MX
```

and normally returns:

```
zabbix.com           MX       0 mail.zabbix.com
```

So our trigger expression to detect if the DNS query result deviated from the expected result is:

```
{Zabbix
server:net.dns.record[8.8.8.8,{$WEBSITE_NAME},{$DNS_RESOURCE_RECORD_TYPE},2,
1].last()}<>"{$WEBSITE_NAME}           {$DNS_RESOURCE_RECORD_TYPE}       0
mail.{$WEBSITE_NAME}"
```

Notice the quotes around the second operand.

Example 16

Comparing two string values - operands are:

- a function that returns a string
- a string constant with special characters \ and "

Problem: detect if the `/tmp/hello` file content is equal to:

```
\" //hello ?\"
```

Option 1) write the string directly

```
{Zabbix server:vfs.file.contents[/tmp/hello].last()}="\\\" //hello ?\\\""
```

Notice how \ and " characters are escaped when the string gets compared directly.

Option 2) use a macro

```
{$HELLO_MACRO} = \" //hello ?\"
```

in the expression:

```
{Zabbix server:vfs.file.contents[/tmp/hello].last()}={$HELLO_MACRO}
```

Hysteresis

Sometimes an interval is needed between problem and recovery states, rather than a simple threshold. For example, if we want to define a trigger that reports a problem when server room temperature goes above 20°C and we want it to stay in the problem state until the temperature drops below 15°C, a simple trigger threshold at 20°C will not be enough.

Instead, we need to define a trigger expression for the problem event first (temperature above 20°C). Then we need to define an additional recovery condition (temperature below 15°C). This is done by defining an additional *Recovery expression* parameter when defining a trigger.

In this case, problem recovery will take place in two steps:

- First, the problem expression (temperature above 20°C) will have to evaluate to FALSE
- Second, the recovery expression (temperature below 15°C) will have to evaluate to TRUE

The recovery expression will be evaluated only when the problem event is resolved first.

The recovery expression being TRUE alone does not resolve a problem if the problem expression is still TRUE!

Example 1

Temperature in server room is too high.

Problem expression:

```
{server:temp.last()}>20
```

Recovery expression:

```
{server:temp.last()}<=15
```

Example 2

Free disk space is too low.

Problem expression: it is less than 10GB for last 5 minutes

```
{server:vfs.fs.size[/,free].max(5m)}<10G
```

Recovery expression: it is more than 40GB for last 10 minutes

```
{server:vfs.fs.size[/,free].min(10m)}>40G
```

Expressions with unsupported items and unknown values

Versions before Zabbix 3.2 are very strict about unsupported items in a trigger expression. Any unsupported item in the expression immediately renders trigger value to Unknown.

Since Zabbix 3.2 there is a more flexible approach to unsupported items by admitting unknown values into expression evaluation:

- For some functions their values are not affected by whether an item is supported or unsupported. Such functions are now evaluated even if they refer to unsupported items. See the list in functions and unsupported items.
- Logical expressions with OR and AND can be evaluated to known values in two cases regardless of unknown operands:
  - "1 `or` Unsuported_item1.some_function() `or` Unsuported_item2.some_function() `or` ..." can be evaluated to '1' (True),
  - "0 `and` Unsuported_item1.some_function() `and` Unsuported_item2.some_function() `and` ..." can be evaluated to '0' (False).
    Zabbix tries to evaluate logical expressions taking unsupported items as Unknown values. In the two cases mentioned above a known value will be produced; in other cases trigger value will be Unknown.
- If a function evaluation for supported item results in error, the function value is Unknown and it takes part in further expression evaluation.

Note that unknown values may "disappear" only in logical expressions as described above. In

arithmetic expressions unknown values always lead to result Unknown (except division by 0).

If a trigger expression with several unsupported items evaluates to Unknown the error message in the frontend refers to the last unsupported item evaluated.