1 User macros

Overview

For greater flexibility, Zabbix supports user macros, which can be defined on global, template and host level. These macros have a special syntax: **{$MACRO}**. An optional context can be used in user macros, allowing to override default value with context specific one. User macros with context have similar syntax: **{$MACRO:context}**.

The macros can be used in:

- item names
- item key parameters
- trigger names and descriptions
- trigger expression parameters and constants (see examples)
- several other locations

The following characters are allowed in the macro names: **A-Z** , **0-9** , **_** , **.**

Zabbix substitutes macros according to the following precedence:

1. host level macros (checked first)
2. macros defined for first level templates of the host (i.e., templates linked directly to the host), sorted by template ID
3. macros defined for second level templates of the host, sorted by template ID
4. macros defined for third level templates of the host, sorted by template ID
5. ...
6. global macros (checked last)

In other words, if a macro does not exist for a host, Zabbix will try to find it in the host templates of increasing depth. If still not found, a global macro will be used, if exists.

If Zabbix is unable to find a macro, the macro will not be substituted.

To define user macros, go to the corresponding locations in the frontend:

- for global macros, visit *Administration → General → Macros*
- for host and template level macros, open host or template properties and look for the *Macros* tab

If a user macro is used in items or triggers in a template, it is suggested to add that macro to the template even if it is defined on a global level. That way, exporting the template to XML and importing it in another system will still allow it to work as expected.

Most common use cases of global and host macros:

1. taking advantage of templates with host specific attributes: passwords, port numbers, file names, regular expressions, etc
2. global macros for global one-click configuration changes and fine tuning

Macro context

Macro context is a text value, quoted with " if context contains } character or starts with " character. Quotes inside quoted macros must be escaped with \ character. The \ character itself is not escaped, which means it's impossible to have quoted macro ending with \ character - the macro {$MACRO:"a:\b\c\"} is invalid.

Only discovery macros are supported in macro contexts. Any other macros are ignored and treated as a text values.

The common use case for macro contexts would be using LLD macro value as a user macro context. For example a trigger prototype could be defined for mounted file system discovery to use different low space limit depending on mount points or file system types.

When lookin up a macro with context the following rules are used:

- the leading spaces in context are ignored, the trailing spaces are not. For example {$MACRO:A} is the same as {$MACRO: A}, but not {$MACRO:A }.
- all spaces before leading quotes and after trailing quotes are ignored, but all spaces inside quotes are not. Macros {$MACRO:"A"}, {$MACRO: "A"}, {$MACRO:"A" } and {$MACRO: "A" } are the same, but macros {$MACRO:"A"} and {$MACRO:" A "} are not.

When context macros are processed the macro with its context is evaluated. If a macro with this context is not defined by host or linked templates, and it is not a defined as global macro with context, then the macro without context is evaluated.

Examples

Example 1

Use of host-level macro in the "Status of SSH daemon" item key:

```
net.tcp.service[ssh,,{$SSH_PORT}]
```

This item can be assigned to multiple hosts, providing that the value of **{$SSH_PORT}** is defined on those hosts.

Example 2

Use of host-level macro in the "CPU load is too high" trigger:

```
{ca_001:system.cpu.load[,avg1].last()}>{$MAX_CPULOAD}
```

Such a trigger would be created on the template, not edited in individual hosts.

If you want to use amount of values as the function parameter (for example, **max(#3)**), include hash mark in the macro definition like this: SOME_PERIOD ⇒ #3

Example 3

Use of two macros in the “CPU load is too high” trigger:

```
{ca_001:system.cpu.load[,avg1].min({$CPULOAD_PERIOD})}>{$MAX_CPULOAD}
```

Note that a macro can be used as a parameter of trigger function, in this example function **min()**.

In trigger expressions user macros will expand if referencing a parameter or constant. They will NOT expand if referencing the host, item key, function, operator or another trigger expression.

Example 4

Set the free disk space trigger prototype for a host to:

```
{Host:vfs.fs.size[{#FSNAME},pfree].last(0)}<{$LOW_SPACE_LIMIT:#FSNAME}
```

Add macros:

- `{$LOW_SPACE_LIMIT}` **10**
- `{$LOW_SPACE_LIMIT:/tmp}` **50**

If, for example, the mounted file system discovery discovered filesystems `/` and `/tmp`, then events will be created when `/` filesystem has less than **10**% of free disk space or `/tmp` filesystem has less than **50**% of free disk space.