2 User macros

Overview

User macros are supported in Zabbix for greater flexibility, in addition to the macros supported out-of-the-box.

User macros can be defined on global, template and host level. These macros have a special syntax:

```
{$MACRO}
```

User macros can be used in:

- item names
- item key parameters
- trigger names and descriptions
- trigger expression parameters and constants (see examples)
- many other locations (see Macros supported by location)

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

Zabbix resolves 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, etc.
5. 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 resolved.

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.

Common use cases of global and host macros

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

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 resolve if referencing a parameter or constant. They will NOT resolve if referencing the host, item key, function, operator or another trigger expression.

User macro context

An optional context can be used in user macros, allowing to override the default value with context-specific one.

User macros with context have a similar syntax:

```
{$MACRO:context}
```

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

Only low-level discovery macros are supported in macro contexts. Any other macros are ignored and

treated as plain text.

Technically, macro context is specified using rules similar to item key parameters, except macro context is not parsed as several parameters if there is a `,` character:

- Macro context must be quoted with `"` if the context contains a `}` character or starts with a `"` character. Quotes inside quoted context must be escaped with the `\` character. The `\` character itself is not escaped, which means it's impossible to have a quoted context ending with the `\` character - the macro `{$MACRO:"a:\b\c\"}` is invalid.
- The leading spaces in context are ignored, the trailing spaces are not. For example `{$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.

The following macros are all equivalent, because they have the same context: `{$MACRO:A}`, `{$MACRO: A}` and `{$MACRO:"A"}`. This is in contrast with item keys, where `key[a]`, `key[ a]` and `key["a"]` are the same semantically, but different for uniqueness purposes.

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

See usage example of macro context in a disk space trigger prototype and take limitation clause into consideration.