12. Regular expressions

Overview

Perl Compatible Regular Expressions (PCRE) are supported in Zabbix.

There are two ways of using regular expressions in Zabbix:

- manually entering a regular expression
- using a global regular expression created in Zabbix

Regular expressions

You may manually enter a regular expression in supported places. Note that the expression may not start with @ because that symbol is used in Zabbix for referencing global regular expressions.

It's possible to run out of stack when using regular expressions. See the pcrestack man page for more information.

Note that in multi-line matching, the ^ and $ anchors match at the beginning/end of each line respectively, instead of the beginning/end of the entire string.

Global regular expressions

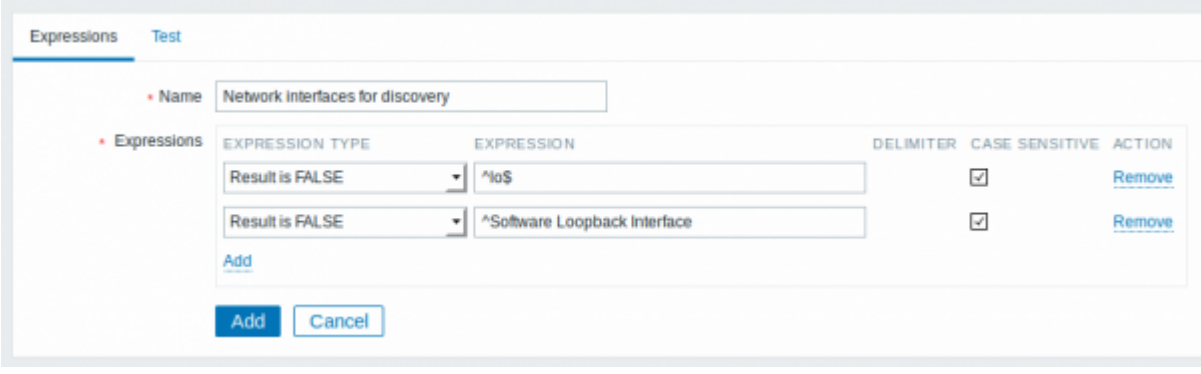
There is an advanced editor for creating and testing complex regular expressions in Zabbix frontend.

Once a regular expression has been created this way, it can be used in several places in the frontend by referring to its name, prefixed with @, for example, *@mycustomregexp*.

To create a global regular expression:

- Go to: *Administration* → *General*
- Select *Regular expressions* from the dropdown
- Click on *New regular expression*

The **Expressions** tab allows to set the regular expression name and add subexpressions.



All mandatory input fields are marked with a red asterisk.

Parameter	Description
Name	Set the regular expression name. Any Unicode characters are allowed.
Expressions	Click on *Add* in the Expressions block to add a new subexpression.
Expression type	Select expression type: **Character string included** - match the substring **Any character string included** - match any substring from a delimited list. The delimited list includes a comma (,), a dot (.) or a forward slash (/). **Character string not included** - match any string except the substring **Result is TRUE** - match the regular expression **Result is FALSE** - do not match the regular expression
Expression	Enter substring/regular expression.
Delimiter	A comma (,), a dot (.) or a forward slash (/) to separate text strings in a regular expression. This parameter is active only when "*Any character string included*" expression type is selected.
Case sensitive	A checkbox to specify whether a regular expression is sensitive to capitalization of letters.

Since Zabbix 2.4.0, a forward slash (/) in the expression is treated literally, rather than a delimiter. This way it is possible to save expressions containing a slash, whereas previously it would produce an error.

A custom regular expression name in Zabbix may contain commas, spaces, etc. In those cases where that may lead to misinterpretation when referencing (for example, a comma in the parameter of an item key) the whole reference may be put in quotes like this: "*@My custom regexp for purpose1, purpose2*".
Regular expression names must not be quoted in other locations (for example, in LLD rule properties).

Default global regular expressions

Zabbix comes with several global regular expression in its default dataset.

Name	Expression	Matches
File systems for discovery	`^(btrfs|ext2|ext3|ext4|jfs|reiser|xfs|ffs|ufs|jfs|jfs2|vxfs|hfs|refs|apfs|ntfs|fat32|zfs)$`	"btrfs" or "ext2" or "ext3" or "ext4" or "jfs" or "reiser" or "xfs" or "ffs" or "ufs" or "jfs" or "jfs2" or "vxfs" or "hfs" or "refs" or "apfs" or "ntfs" or "fat32" or "zfs"
Network interfaces for discovery	`^Software Loopback Interface`	Strings starting with "Software Loopback Interface".
	`^lo$`	"lo"
	`^(In)?[Ll]oop[Bb]ack[0-9._]*$`	Strings that optionally start with "In", then have "L" or "l", then "oop", then "B" or "b", then "ack", which can be optionally followed by any number of digits, dots or underscores.
	`^NULL[0-9.]*$`	Strings starting with "NULL" optionally followed by any number of digits or dots.
	`^[Ll]o[0-9.]*$`	Strings starting with "Lo" or "lo" and optionally followed by any number of digits or dots.
	`^[Ss]ystem$`	"System" or "system"
	`^Nu[0-9.]*$`	Strings starting with "Nu" optionally followed by any number of digits or dots.

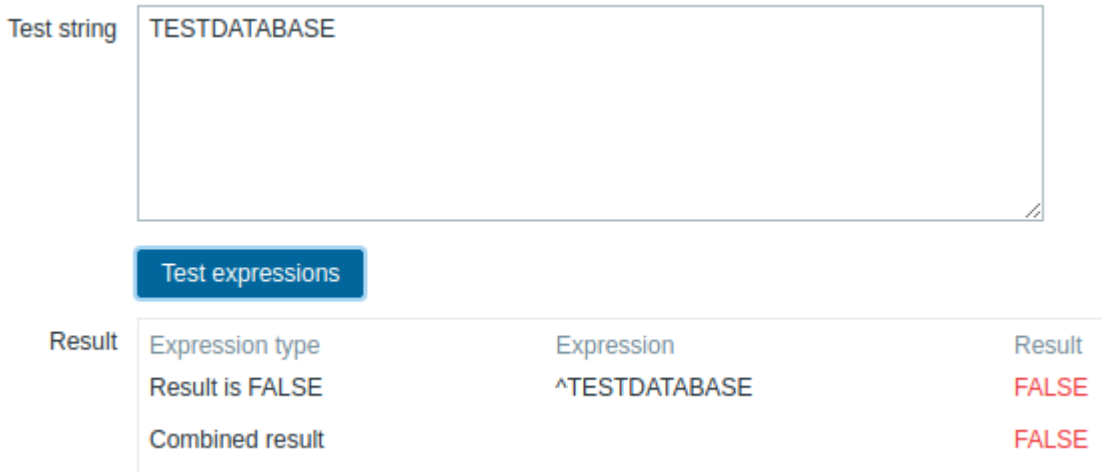
Name	Expression	Matches
Storage devices for SNMP discovery	`^(Physical memory|Virtual memory|Memory buffers|Cached memory|Swap space)$`	"Physical memory" or "Virtual memory" or "Memory buffers" or "Cached memory" or "Swap space"
Windows service names for discovery	`^(MMCSS|gupdate|SysmonLog|clr_optimization_v2.0.50727_32|clr_optimization_v4.0.30319_32)$`	"MMCSS" or "gupdate" or "SysmonLog" or strings like "clr_optimization_v2.0.50727_32" and "clr_optimization_v4.0.30319_32" where instead of dots you can put any character except newline.
Windows service startup states for discovery	`^(automatic|automatic delayed)$`	"automatic" or "automatic delayed"

Examples

Example 1

Use of the following expression in low-level discovery to discover databases except a database with a specific name:

```
^TESTDATABASE$
```

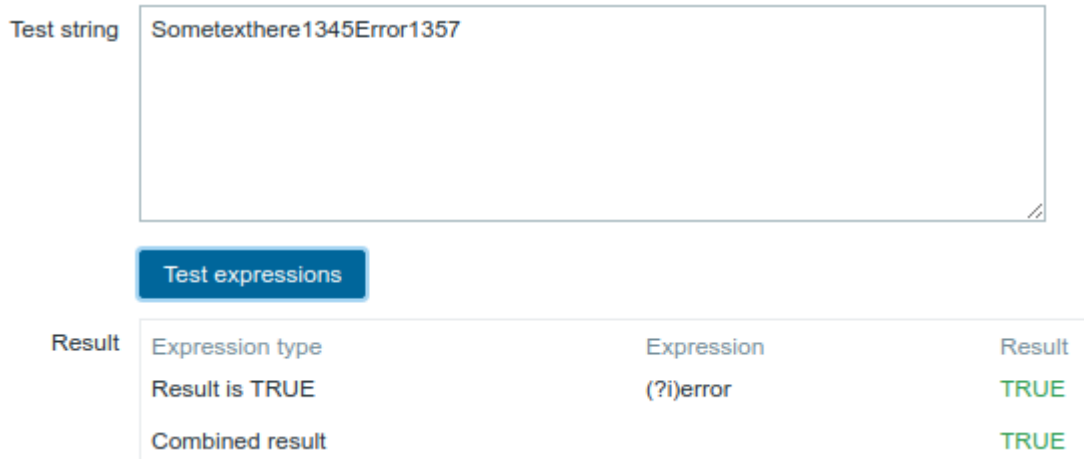


Chosen *Expression type*: "Result is FALSE". Doesn't match name, containing string "*TESTDATABASE*".

Example with an inline regex modifier

Use of the following regular expression including an inline modifier (?i) to match the characters "error":

```
(?i)error
```



Chosen *Expression type*: "Result is TRUE". Characters “error” are matched.

Another example with an inline regex modifier

Use of the following regular expression including multiple inline modifiers to match the characters after a specific line:

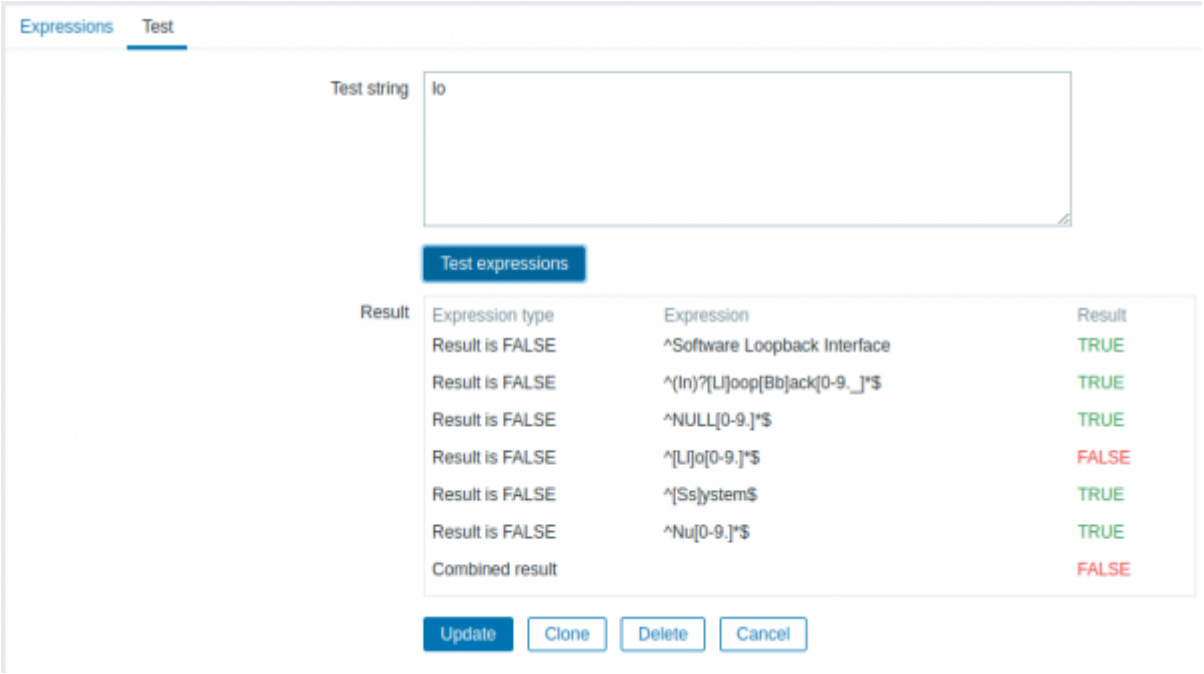
```
(?<=match (?i)everything(?-i) after this line\n)(?sx).*# we add s modifier to allow . match newline characters
```

Chosen Expression type: "Result is TRUE". Characters after a specific line are matched.

g modifier can't be specified in line. The list of available modifiers can be found in pcresyntax man page. For more information about PCRE syntax please refer to PCRE HTML documentation.

More complex example

A custom regular expression may consist of multiple subexpressions, and it can be tested in the **Test** tab by providing a test string.



Results show the status of each subexpression and total custom expression status.

Total custom expression status is defined as *Combined result*. If several sub expressions are defined Zabbix uses AND logical operator to calculate *Combined result*. It means that if at least one Result is False *Combined result* has also False status.

Regular expression support by location

Location	Regular expression	Global regular expression	Comments
Agent items			

Location		Regular expression	Global regular expression	Comments
	eventlog[]	Yes	Yes	`regexp`, `severity`, `source`, `eventid` parameters
	log[]			`regexp` parameter
	log.count[]			
	logrt[]		Yes/No	`regexp` parameter supports both, `file_regexp` parameter supports non-global expressions only
	logrt.count[]			
	proc.cpu.util[]		No	`cmdline` parameter
	proc.mem[]			
	proc.num[]			
	sensor[]			`device` and `sensor` parameters on Linux 2.4
	system.hw.macaddr[]			`interface` parameter
	system.sw.packages[]			`package` parameter
	vfs.dir.count[]			`regex_incl` and `regex_excl` parameters
	vfs.dir.size[]			`regex_incl` and `regex_excl` parameters
	vfs.file.regexp[]			`regexp` parameter
	vfs.file.regmatch[]			
	web.page.regexp[]			
SNMP traps				
	snmptrap[]	Yes	Yes	`regexp` parameter
Item value preprocessing		Yes	No	`pattern` parameter
Trigger functions				
	count()	Yes	Yes	`pattern` parameter if `operator` parameter is *regexp* or *iregexp*
	logeventid()			`pattern` parameter
	logsource()			
	iregexp()			
	regexp()			
Low-level discovery		Yes	Yes	*Filter* field
Action conditions		Yes	No	In *matches*, *does not match* options for *Host name* and *Host metadata* autoregistration conditions
Web monitoring		Yes	No	*Variables* with a **regex:** prefix *Required string* field
User macro context		Yes	No	In macro context with a **regex:** prefix Supported since Zabbix 5.0.2.
Macro functions				
	regsub()	Yes	No	`pattern` parameter
	iregsub()			
Icon mapping		Yes	Yes	*Expression* field