5.3 Low-level discovery

Overview

Low-level discovery provides a way to automatically create items, triggers, and graphs for different entities on a computer. For instance, Zabbix can automatically start monitoring file systems or network interfaces on your machine, without the need to create items for each file system or network interface manually. Additionally it is possible to configure Zabbix to remove unneeded entities automatically based on actual results of periodically performed discovery.

In Zabbix, four types of item discovery are supported out of the box:

- discovery of file systems;
- discovery of network interfaces;
- discovery of CPUs and CPU cores;
- discovery of SNMP OIDs.

A user can define their own types of discovery, provided they follow a particular JSON protocol.

The general architecture of the discovery process is as follows.

First, a user creates a discovery rule in “Configuration” → “Templates” → “Discovery” column. A discovery rule consists of (1) an item that discovers the necessary entities (for instance, file systems or network interfaces) and (2) prototypes of items, triggers, and graphs that should be created based on the value of that item.

An item that discovers the necessary entities is like a regular item seen elsewhere: the server asks a Zabbix agent (or whatever the type of the item is set to) for a value of that item, the agent responds with a textual value. The difference is that the value the agent responds with should contain a list of discovered entities in a specific JSON format. While the details of this format are only important for implementers of custom discovery checks, it is necessary to know that the returned value contains a list of macro → value pairs. For instance, item “net.if.discovery” might return two pairs: “{#IFNAME}” → “lo” and “{#IFNAME}” → “eth0”.

Low-level discovery items “vfs.fs.discovery” and “net.if.discovery” are supported since Zabbix agent version 2.0.
Discovery item “system.cpu.discovery” is supported since Zabbix agent version 2.4.
Discovery of SNMP OIDs is supported since Zabbix server and proxy version 2.0.
Return values of a low-level discovery rule are limited to 2048 bytes on a Zabbix proxy run with IBM DB2 database. This limit does not apply to Zabbix server as return values are processed without being stored in a database.

These macros are used in names, keys and other prototype fields where they are then substituted with the received values for creating real items, triggers, graphs or even hosts for each discovered entity. See the full list of options for using LLD macros.

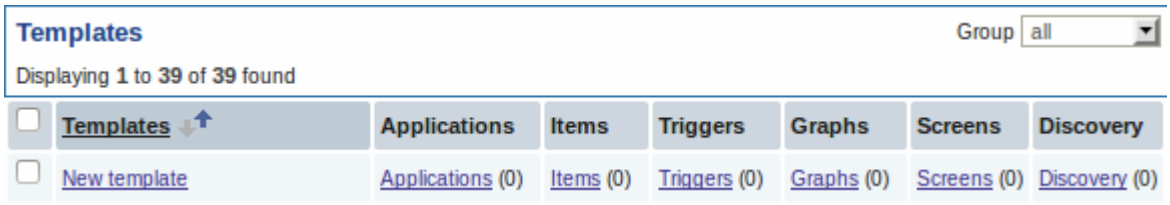
When the server receives a value for a discovery item, it looks at the macro → value pairs and for each pair generates real items, triggers, and graphs, based on their prototypes. In the example with “net.if.discovery” above, the server would generate one set of items, triggers, and graphs for the loopback interface “lo”, and another set for interface “eth0”.

The following sections illustrate the process described above in detail and serve as a how-to for performing discovery of file systems, network interfaces, and SNMP OIDs. The last section describes the JSON format for discovery items and gives an example of how to implement your own file system discoverer as a Perl script.

5.3.1 Discovery of file systems

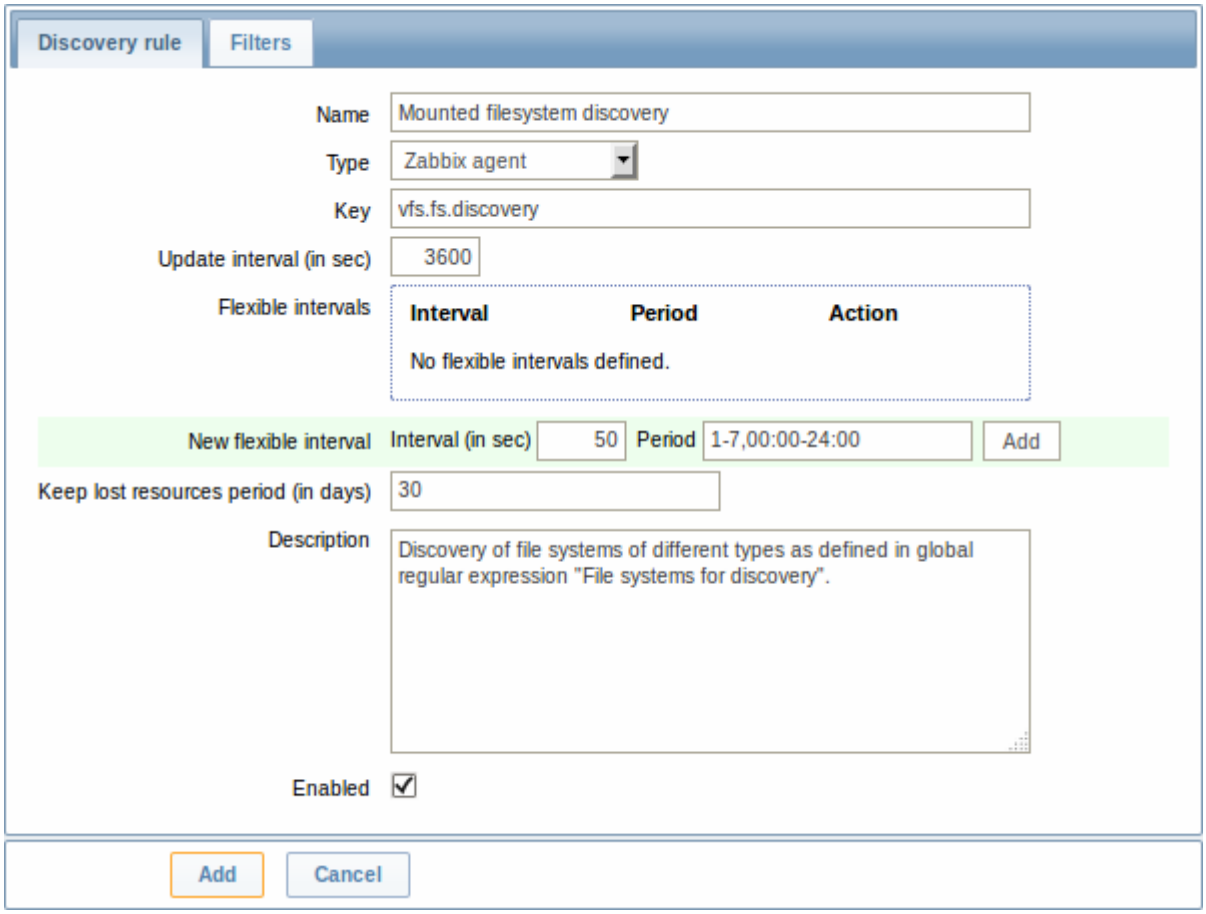
To configure the discovery of file systems, do the following:

- Go to: *Configuration* → *Templates*
- Click on *Discovery* in the row of an appropriate template



- Click on *Create discovery rule* in the upper right corner of the screen
- Fill in the form with the following details

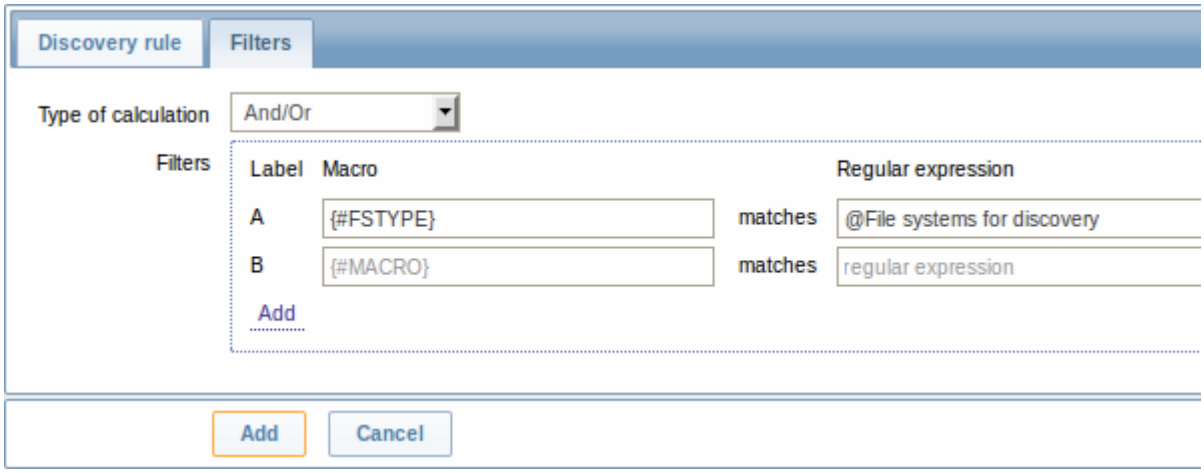
The **Discovery rule** tab contains general discovery rule attributes:



Parameter	Description
Name	Name of discovery rule.

Parameter	Description
Type	The type of check to perform discovery; should be *Zabbix agent* or *Zabbix agent (active)* for file system discovery.
Key	An item with “vfs.fs.discovery” key is built into the Zabbix agent on many platforms (see supported item key list for details), and will return a JSON with the list of file systems present on the computer and their types.
Update interval (in sec)	This field specifies how often Zabbix performs discovery. In the beginning, when you are just setting up file system discovery, you might wish to set it to a small interval, but once you know it works you can set it to 30 minutes or more, because file systems usually do not change very often. *Note*: If set to '0', the item will not be polled. However, if a flexible interval also exists with a non-zero value, the item will be polled during the flexible interval duration.
Flexible intervals	You can create exceptions to *Update interval*. For example: Interval: **0**, Period: **6-7,00:00-24:00** - will disable the polling at the weekend. Otherwise default update interval will be used. Up to seven flexible intervals can be defined. If multiple flexible intervals overlap, the smallest *Interval* value is used for the overlapping period. See Time period specification page for description of the *Period* format. *Note*: If set to '0', the item will not be polled during the flexible interval duration and will resume polling according to the *Update interval* once the flexible interval period is over.
Keep lost resources period (in days)	This field allows you to specify for how many days the discovered entity will be retained (won't be deleted) once its discovery status becomes “Not discovered anymore” (max 3650 days). *Note:* If set to “0”, entities will be deleted immediately. Using “0” is not recommended, since just wrongly editing the filter may end up in the entity being deleted with all the historical data.
Description	Enter a description.
Enabled	If checked, the rule will be processed.

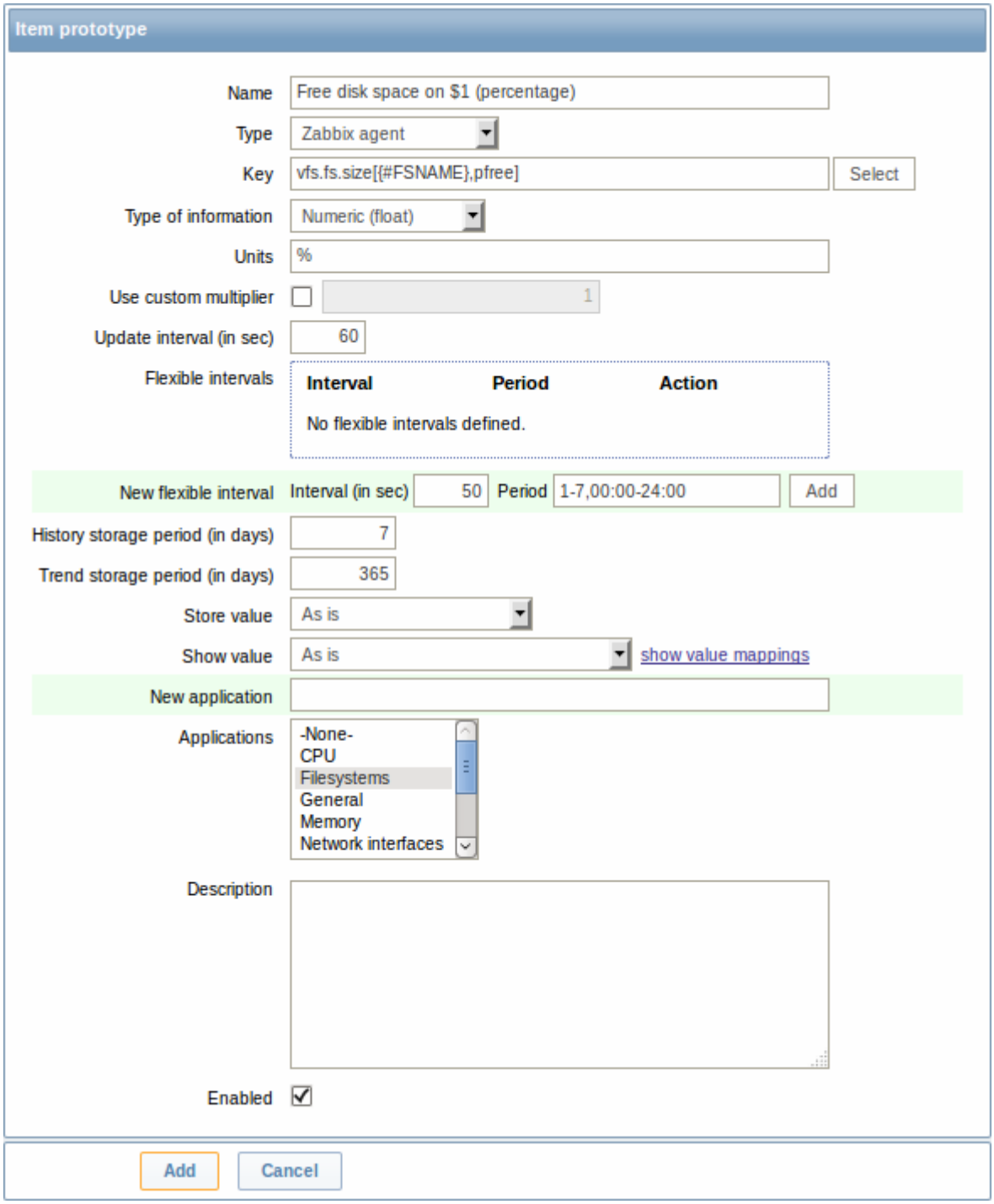
The **Filters** tab contains discovery rule filter definitions:



Parameter	Description
Type of calculation	The following options for calculating filters are available: **And** - all filters must be passed; **Or** - enough if one filter is passed; **And/Or** - uses *And* with different macro names and *Or* with the same macro name; **Custom expression** - a user-defined formula for calculating filters. The formula must include all filters in the list (represented as uppercase letters A, B, C, ...) and may include spaces, tabs, brackets ( ), **and** (case sensitive), **or** (case sensitive). Limited to 255 symbols. The custom calculation is the same as in action conditions.
Filters	A filter can be used to generate only real items, triggers, and graphs for certain file systems. It expects a POSIX Extended Regular Expression. For instance, if you are only interested in C:, D:, and E: file systems, you could put {#FSNAME} into “Macro” and "^C|^D|^E" regular expression into “Regular expression” text fields. Filtering is also possible by file system types using {#FSTYPE} macro (e. g. "^ext|^reiserfs"). You can enter a regular expression or reference a global regular expression in “Regular expression” field. In order to test a regular expression you can use “grep -E”, for example: `for f in ext2 nfs reiserfs smbfs; do echo $f | grep -E '^ext|^reiserfs' || echo "SKIP: $f"; done` Defining several filters is supported since Zabbix **2.4.0**. Note that if some macro from the filter is missing in the response, the found entity will be ignored.

Zabbix database in MySQL must be created as case-sensitive if file system names that differ only by case are to be discovered correctly.
Discovery rule history is not preserved.

Once a rule is created, go to the items for that rule and press “Create prototype” to create an item prototype. Note how macro {#FSNAME} is used where a file system name is required. When the discovery rule is processed, this macro will be substituted with the discovered file system.



If an item prototype is created with a *Disabled* status, it will be added to a discovered entity, but in a disabled state.

We can create several item prototypes for each file system metric we are interested in:

Item prototypes of Mounted filesystem discovery

Displaying **1** to **5** of **5** found

« Template list **Template:** Template OS Linux « Discovery list **Discovery:** Mounted filesystem discovery Item pr

Trigger prototypes (2) Graph prototypes (1) Host prototypes (0)

Name	Key	Interval	History	Trends	Type
Free disk space on {#FSNAME}	vfs.fs.size[{#FSNAME},free]	60	7	365	Zabbix agent
Free disk space on {#FSNAME} (percentage)	vfs.fs.size[{#FSNAME},pfree]	60	7	365	Zabbix agent
Free inodes on {#FSNAME} (percentage)	vfs.fs.inode[{#FSNAME},pfree]	60	7	365	Zabbix agent
Total disk space on {#FSNAME}	vfs.fs.size[{#FSNAME},total]	3600	7	365	Zabbix agent
Used disk space on {#FSNAME}	vfs.fs.size[{#FSNAME},used]	60	7	365	Zabbix agent

Then, we create trigger prototypes in a similar way:

Trigger prototype

Name: Free disk space is less than 20% on volume {#FSNAME}

Expression: {Template OS Linux:vfs.fs.size[{#FSNAME},pfree].last(0)}<20 Add

Expression constructor

Multiple PROBLEM events generation

Description

URL

Severity: Not classified | Information | **Warning** | Average | High | Disaster

Enabled

Add Cancel

Trigger prototypes of Mounted filesystem discovery

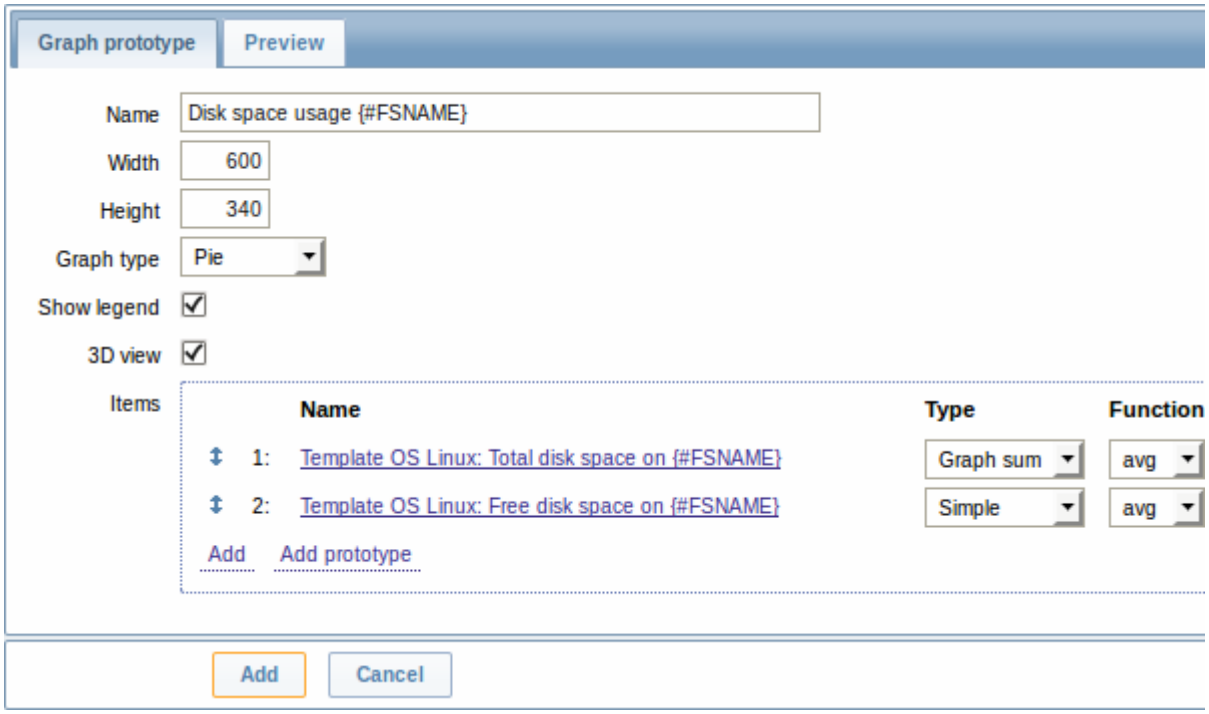
Displaying **1** to **2** of **2** found [Hide disabled

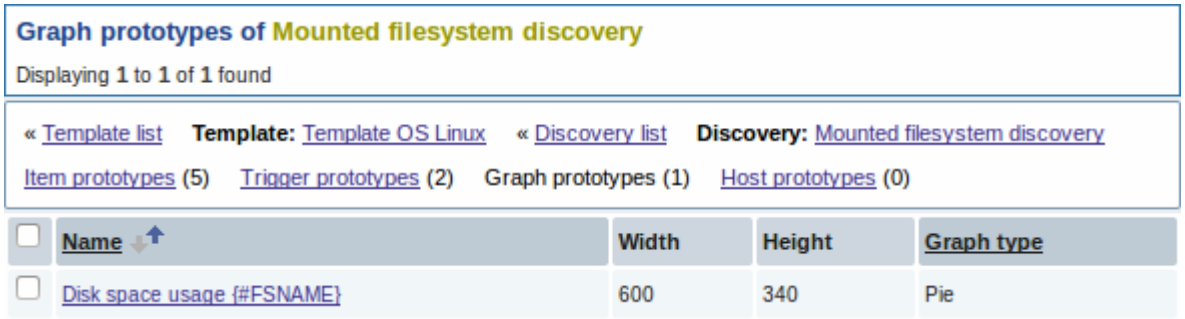
« Template list **Template:** Template OS Linux « Discovery list **Discovery:** Mounted filesystem discovery Item pr

Trigger prototypes (2) Graph prototypes (1) Host prototypes (0)

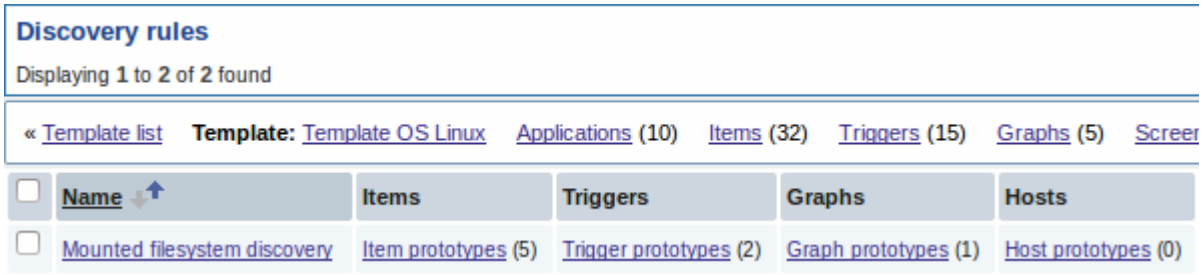
Severity	Name	Expression
Warning	Free disk space is less than 20% on volume {#FSNAME}	{Template OS Linux:vfs.fs.size[{#FSNAME},pfree].last
Warning	Free inodes is less than 20% on volume {#FSNAME}	{Template OS Linux:vfs.fs.inode[{#FSNAME},pfree].la

And graph prototypes too:



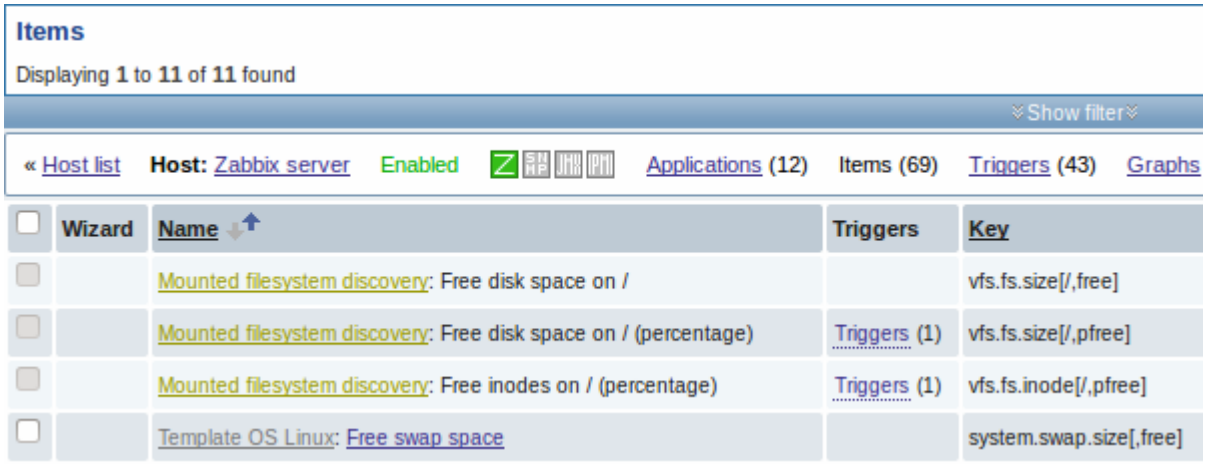


Finally, we have created a discovery rule that looks like shown below. It has five item prototypes, two trigger prototypes, and one graph prototype.



Note: For configuring host prototypes, see the section about host prototype configuration in virtual machine monitoring.

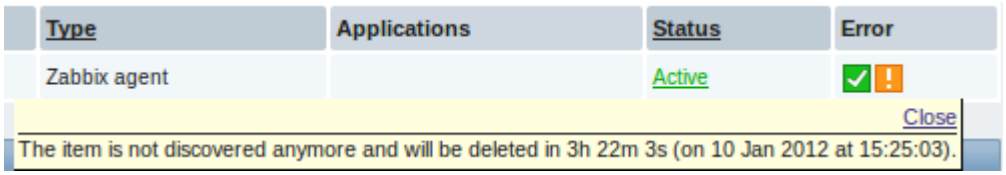
The screenshots below illustrate how discovered items, triggers, and graphs look like in the host's configuration. Discovered entities are prefixed with a golden link to a discovery rule they come from.



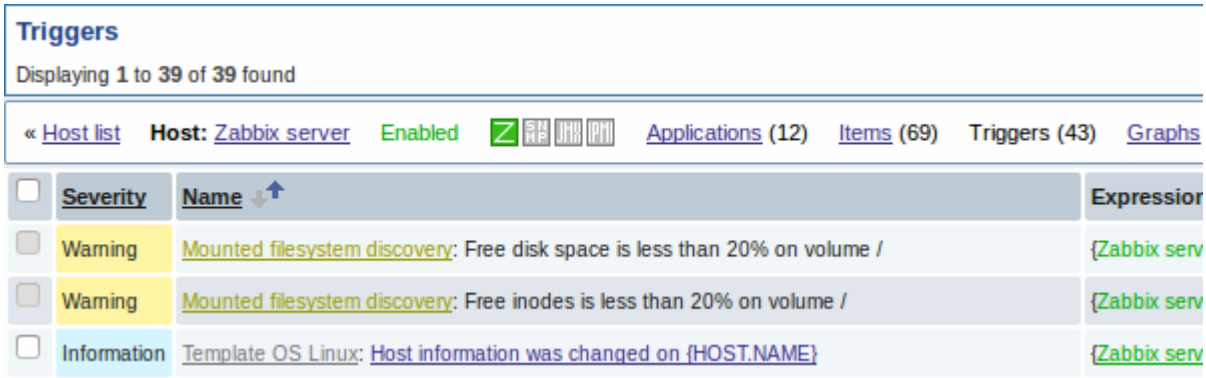
Note that discovered entities will not be created in case there are already existing entities with the same uniqueness criteria, for example, an item with the same key or graph with the same name.

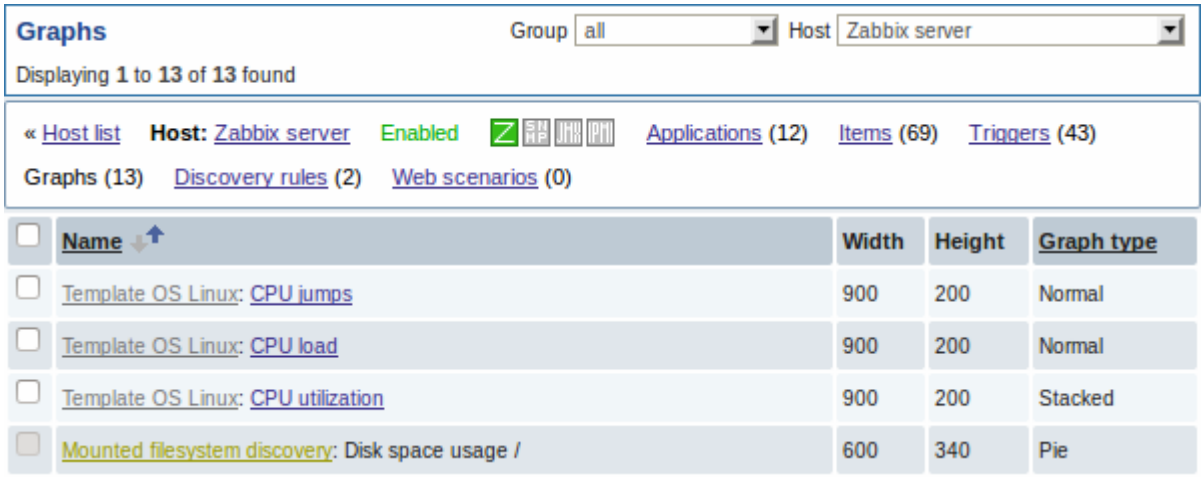
Items (similarly, triggers and graphs) created by a low-level discovery rule cannot be manually deleted. However, they will be deleted automatically if a discovered entity (file system, interface, etc) stops being discovered (or does not pass the filter anymore). In this case the items, triggers and graphs will be deleted after the days defined in the *Keep lost resources period* field pass.

When discovered entities become 'Not discovered anymore', an orange lifetime indicator is displayed in the item list. Move your mouse pointer over it and a message will be displayed indicating how many days are left until the item is deleted.



If entities were marked for deletion, but were not deleted at the expected time (disabled discovery rule or item host), they will be deleted the next time the discovery rule is processed.





5.3.2 Discovery of network interfaces

Discovery of network interfaces is done in exactly the same way as discovery of file systems, except that you use the discovery rule key “net.if.discovery” instead of “vfs.fs.discovery” and use macro {#IFNAME} instead of {#FSNAME} in filter and item/trigger/graph prototypes.

Examples of item prototypes that you might wish to create based on “net.if.discovery”: “net.if.in[{#IFNAME},bytes]”, “net.if.out[{#IFNAME},bytes]”.

See above for more information about the filter.

5.3.3 Discovery of CPUs and CPU cores

Discovery of CPUs and CPU cores is done in a similar fashion as network interface discovery with the exception being that the discovery rule key is “system.cpu.discovery”. This discovery key returns two macros - {#CPU.NUMBER} and {#CPU.STATUS} identifying the CPU order number and status respectively. To note, a clear distinction cannot be made between actual, physical processors, cores and hyperthreads. {#CPU.STATUS} on Linux, UNIX and BSD systems returns the status of the processor, which can be either “online” or “offline”. On Windows systems, this same macro may represent a third value - “unknown” - which indicates that a processor has been detected, but no information has been collected for it yet.

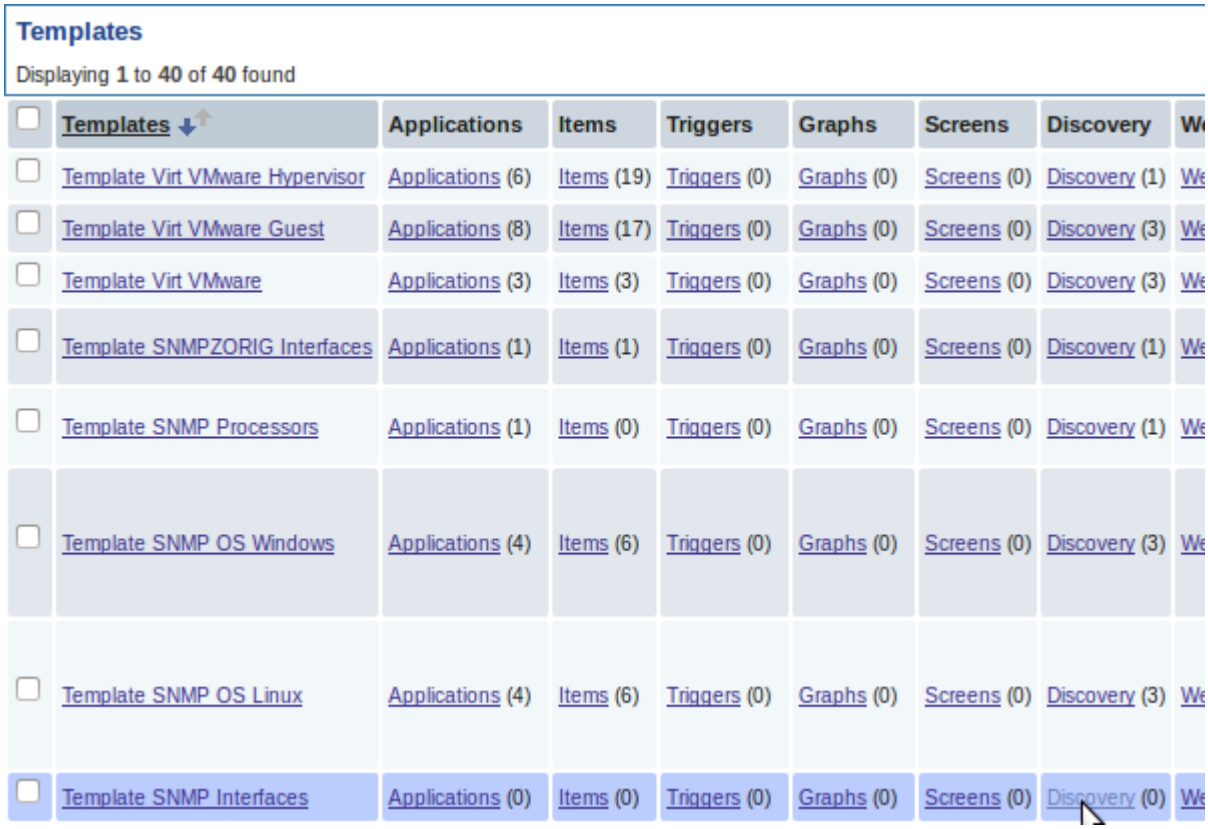
CPU discovery relies on the agent's collector process to remain consistent with the data provided by the collector and save resources on obtaining the data. This has the effect of this item key not working with the test (-t) command line flag of the agent binary, which will return a NOT_SUPPORTED status and an accompanying message indicating that the collector process has not been started.

Item prototypes that can be created based on CPU discovery include, for example, “system.cpu.util[{#CPU.NUMBER}, <type>, <mode>]” or “system.hw.cpu[{#CPU.NUMBER}, <info>]”.

5.3.4 Discovery of SNMP OIDs

In this example, we will perform SNMP discovery on a switch. First, go to “Configuration” →

"Templates".



Templates

Displaying 1 to 40 of 40 found

Templates	Applications	Items	Triggers	Graphs	Screens	Discovery	We
Template Virt VMware Hypervisor	Applications (6)	Items (19)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (1)	We
Template Virt VMware Guest	Applications (8)	Items (17)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (3)	We
Template Virt VMware	Applications (3)	Items (3)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (3)	We
Template SNMPZORIG Interfaces	Applications (1)	Items (1)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (1)	We
Template SNMP Processors	Applications (1)	Items (0)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (1)	We
Template SNMP OS Windows	Applications (4)	Items (6)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (3)	We
Template SNMP OS Linux	Applications (4)	Items (6)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (3)	We
Template SNMP Interfaces	Applications (0)	Items (0)	Triggers (0)	Graphs (0)	Screens (0)	Discovery (0)	We

To edit discovery rules for a template, click on the link in the "Discovery" column.

Then, press "Create rule" and fill the form with the details in the screenshot below.

Unlike file system and network interface discovery, the item does not necessarily have to have "snmp.discovery" key - item type of SNMP agent is sufficient.

Also, unlike the previous examples, this discovery item will generate two macros for each discovered entity: {#SNMPINDEX} and {#SNMPVALUE}. In case you would like to filter out loopback interfaces from returned values you could put "{#SNMPVALUE}" into filter "Macro" and "^([^l]|l$)[^o]?" regular expression into "Regexp" text fields. See above for more information about the filter.

In "SNMP OID" field, we have to put an OID that is capable of generating meaningful values for these macros.

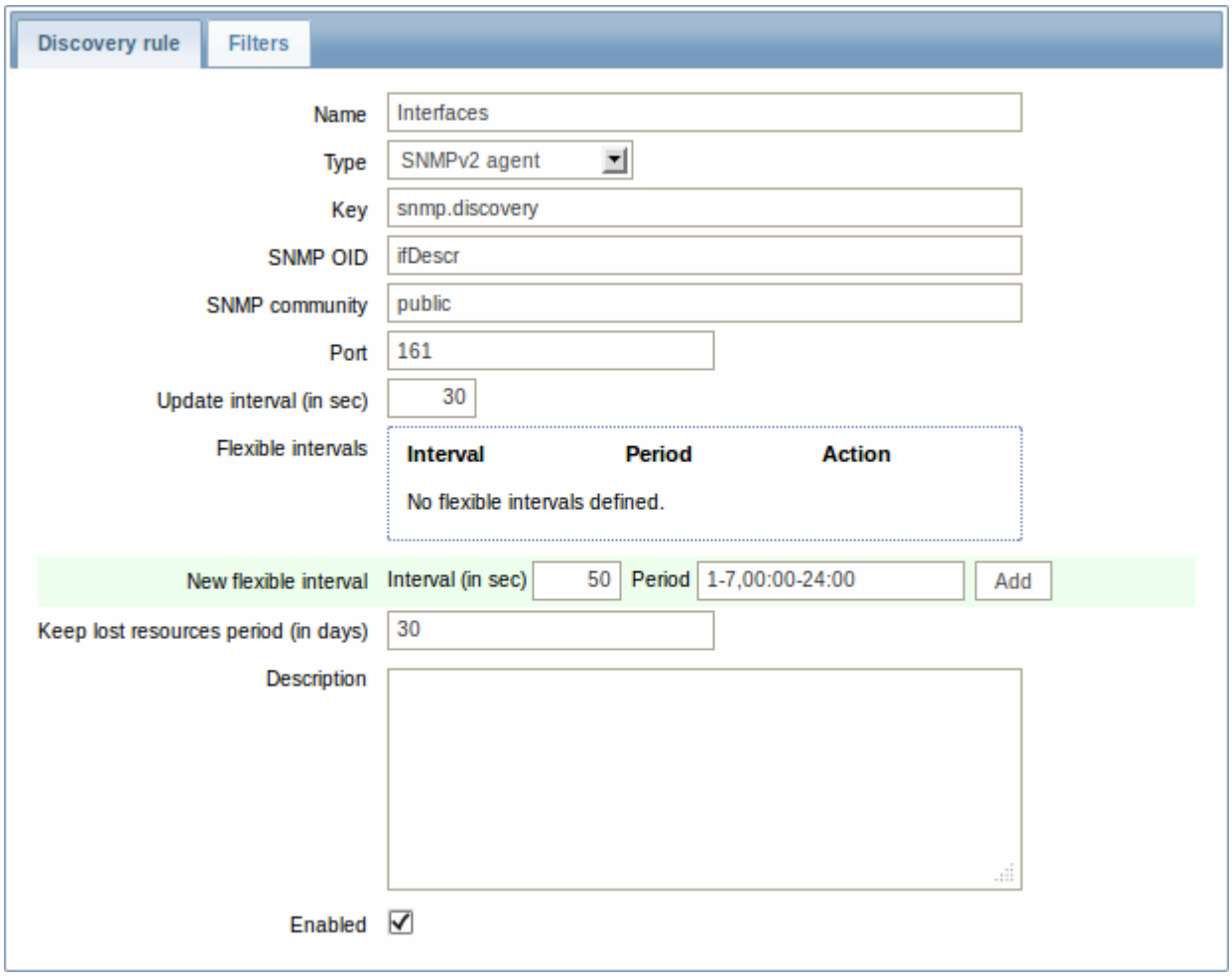
To understand what we mean, let us perform snmpwalk on our switch:

```
$ snmpwalk -v 2c -c public 192.168.1.1 IF-MIB::ifDescr
IF-MIB::ifDescr.1 = STRING: WAN
IF-MIB::ifDescr.2 = STRING: LAN1
IF-MIB::ifDescr.3 = STRING: LAN2
```

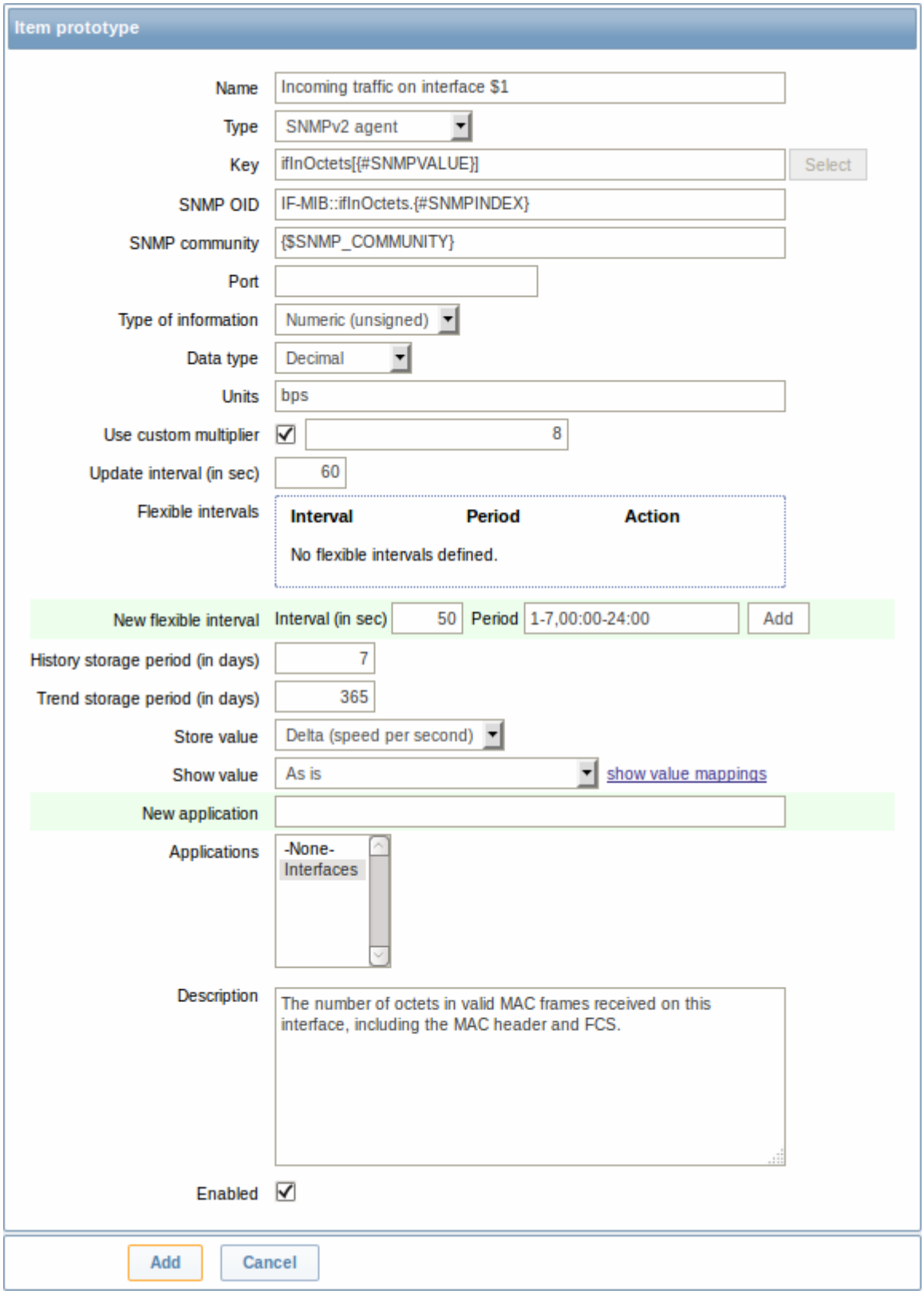
Macro {#SNMPINDEX} takes its value from the part of the OID that is after ifDescr (in this example: 1, 2, 3). Macro {#SNMPVALUE} comes from the value of the corresponding OID (here: WAN, LAN1, LAN2). Thus, our "snmp.discovery" item would return three sets of macro → value pairs:

```
{#SNMPINDEX} -> 1   {#SNMPVALUE} -> WAN
```

```
{#SNMPINDEX} -> 2   {#SNMPVALUE} -> LAN1
{#SNMPINDEX} -> 3   {#SNMPVALUE} -> LAN2
```



The following screenshot illustrates how we can use these macros in item prototypes:



Again, creating as many item prototypes as needed:

Item prototypes of Network interfaces

Displaying **1** to **8** of **8** found

« Template list **Template:** Template SNMP Interfaces « Discovery list **Discovery:** Network interfaces Item proto

Trigger prototypes (1) Graph prototypes (1) Host prototypes (0)

Name	Key	Interval	History	Trends	Type
Admin status of interface {#SNMPVALUE}	ifAdminStatus[{#SNMPVALUE}]	60	7	365	SNMPv2
Alias of interface {#SNMPVALUE}	ifAlias[{#SNMPVALUE}]	3600	7		SNMPv2
Description of interface {#SNMPVALUE}	ifDescr[{#SNMPVALUE}]	3600	7		SNMPv2
Inbound errors on interface {#SNMPVALUE}	ifInErrors[{#SNMPVALUE}]	60	7	365	SNMPv2
Incoming traffic on interface {#SNMPVALUE}	ifInOctets[{#SNMPVALUE}]	60	7	365	SNMPv2
Operational status of interface {#SNMPVALUE}	ifOperStatus[{#SNMPVALUE}]	60	7	365	SNMPv2
Outbound errors on interface {#SNMPVALUE}	ifOutErrors[{#SNMPVALUE}]	60	7	365	SNMPv2
Outgoing traffic on interface {#SNMPVALUE}	ifOutOctets[{#SNMPVALUE}]	60	7	365	SNMPv2

As well as trigger prototypes:

Trigger prototype

Name: Operational status was changed on {HOST.NAME} interface {#SNMI

Expression: {Template SNMP Interfaces:ifOperStatus[{#SNMPVALUE}].diff(0)}=1 Add

Expression constructor

Multiple PROBLEM events generation:

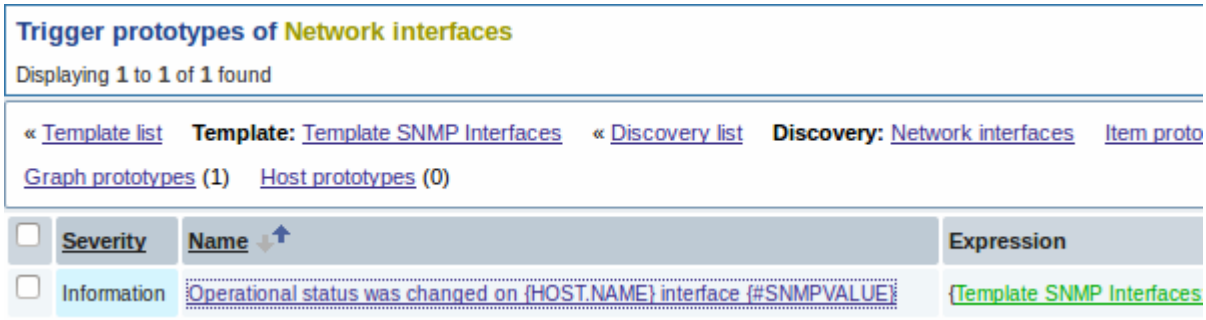
Description:

URL:

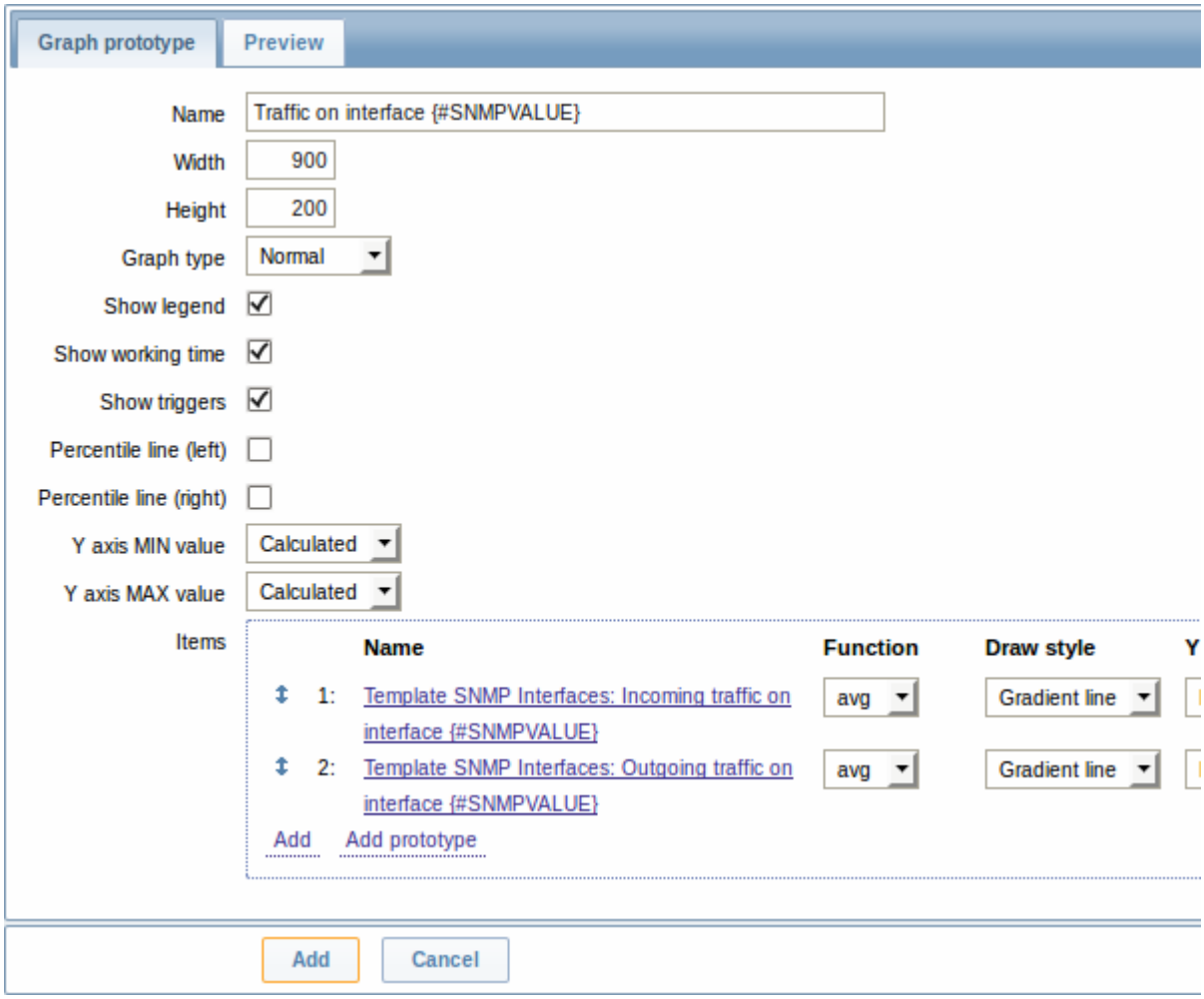
Severity: Not classified | Information | Warning | Average | High | Disaster

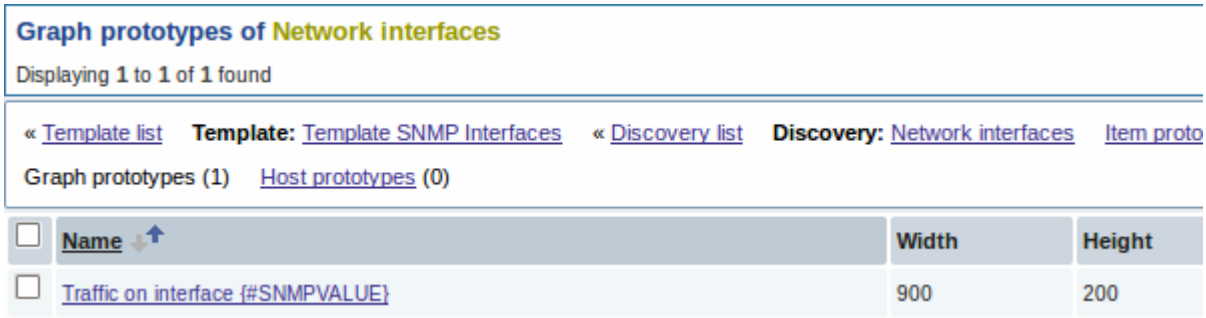
Enabled:

Add Cancel

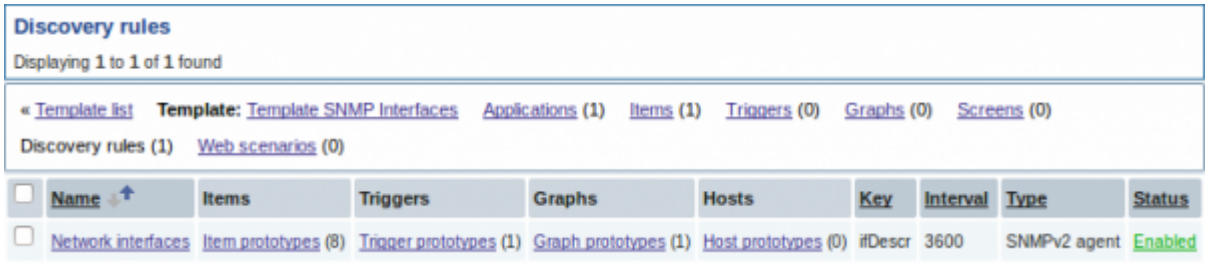


And graph prototypes:





A summary of our discovery rule:



Discovery rules

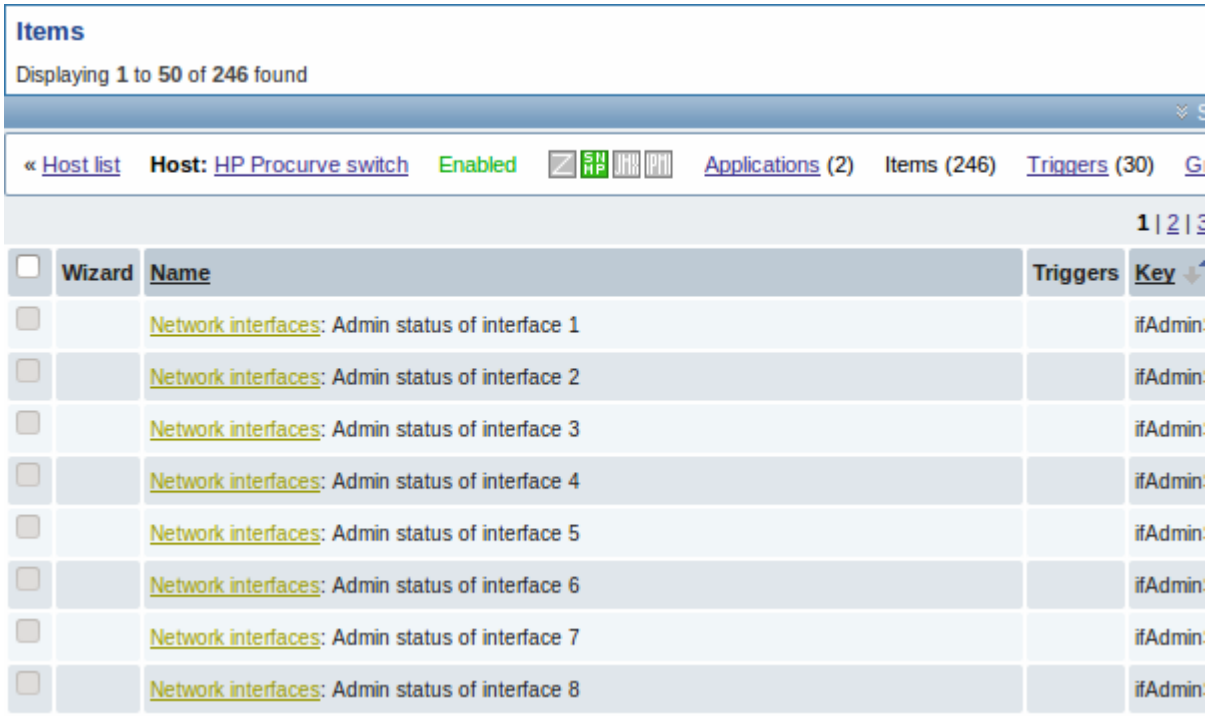
Displaying 1 to 1 of 1 found

« Template list **Template:** Template SNMP Interfaces Applications (1) Items (1) Triggers (0) Graphs (0) Screens (0)

Discovery rules (1) Web scenarios (0)

Name	Items	Triggers	Graphs	Hosts	Key	Interval	Type	Status
Network interfaces	Item prototypes (8)	Trigger prototypes (1)	Graph prototypes (1)	Host prototypes (0)	ifDescr	3600	SNMPv2 agent	Enabled

When server runs, it will create real items, triggers, and graphs, based on the values “snmp.discovery” returns. In host's configuration they will be prefixed with a golden link to a discovery rule they come from.



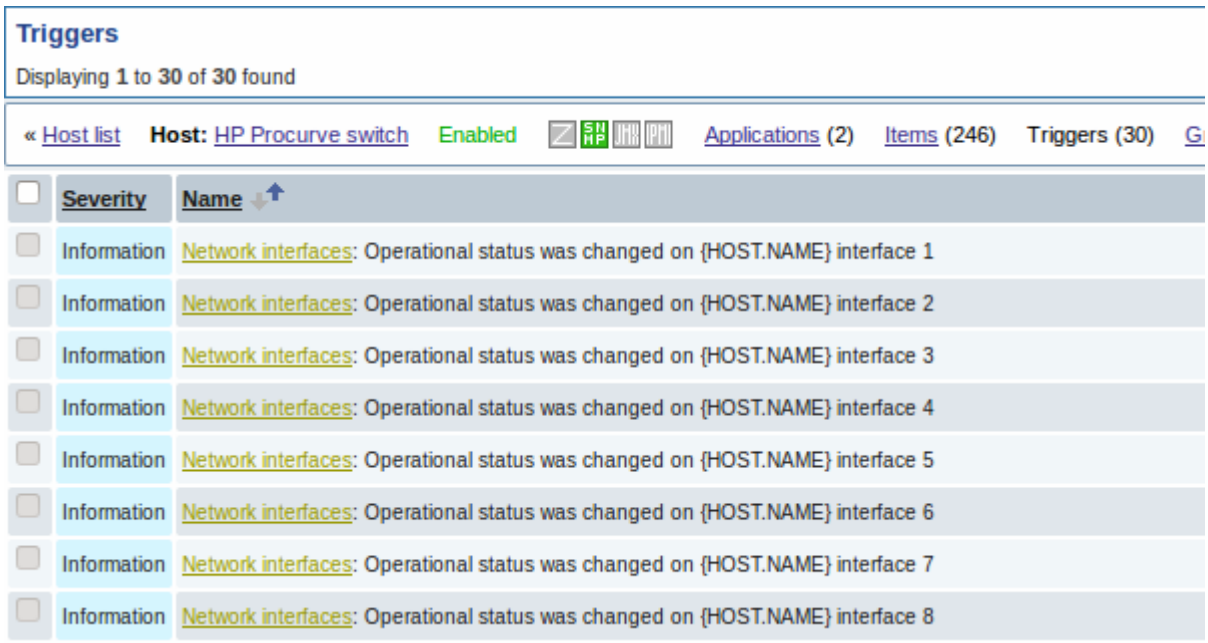
Items

Displaying 1 to 50 of 246 found

« Host list **Host:** HP Procurve switch Enabled Applications (2) Items (246) Triggers (30) G

1 | 2 | 3

Wizard	Name	Triggers	Key
	Network interfaces: Admin status of interface 1		ifAdmin
	Network interfaces: Admin status of interface 2		ifAdmin
	Network interfaces: Admin status of interface 3		ifAdmin
	Network interfaces: Admin status of interface 4		ifAdmin
	Network interfaces: Admin status of interface 5		ifAdmin
	Network interfaces: Admin status of interface 6		ifAdmin
	Network interfaces: Admin status of interface 7		ifAdmin
	Network interfaces: Admin status of interface 8		ifAdmin

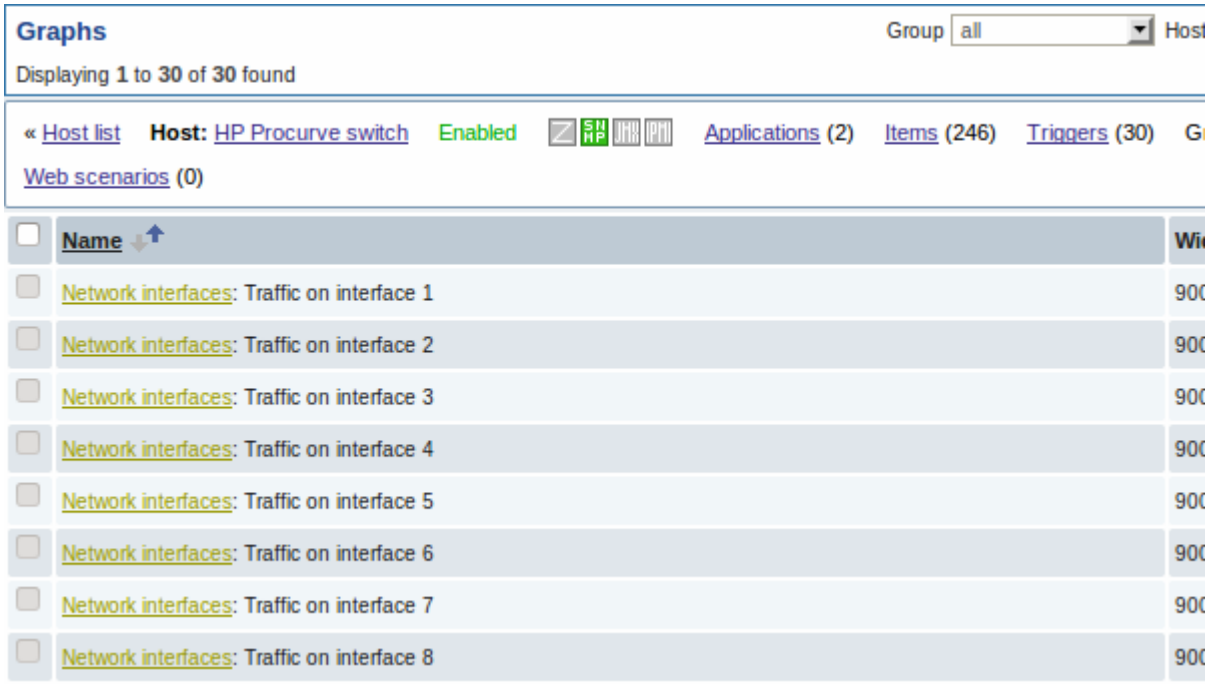


Triggers

Displaying 1 to 30 of 30 found

« Host list **Host:** HP Procurve switch Enabled Applications (2) Items (246) Triggers (30) G

Severity	Name
Information	Network interfaces: Operational status was changed on {HOST.NAME} interface 1
Information	Network interfaces: Operational status was changed on {HOST.NAME} interface 2
Information	Network interfaces: Operational status was changed on {HOST.NAME} interface 3
Information	Network interfaces: Operational status was changed on {HOST.NAME} interface 4
Information	Network interfaces: Operational status was changed on {HOST.NAME} interface 5
Information	Network interfaces: Operational status was changed on {HOST.NAME} interface 6
Information	Network interfaces: Operational status was changed on {HOST.NAME} interface 7
Information	Network interfaces: Operational status was changed on {HOST.NAME} interface 8



5.3.5 Creating custom LLD rules

It is also possible to create a completely custom LLD rule, discovering any type of entities - for example, databases on a database server.

To do so, a custom item should be created that returns JSON, specifying found objects and optionally - some properties of them. The amount of macros per entity is not limited - while the built-in discovery rules return either one or two macros (for example, two for filesystem discovery), it is possible to return more.

The required JSON format is best illustrated with an example. Suppose we are running an old Zabbix 1.8 agent (one that does not support “vfs.fs.discovery”), but we still need to discover file systems. Here is a simple Perl script for Linux that discovers mounted file systems and outputs JSON, which includes both file system name and type. One way to use it would be as a UserParameter with key “vfs.fs.discovery_perl”:

```
#!/usr/bin/perl

$first = 1;

print "{\n";
print "\t\"data\":[\n\n";

for (`cat /proc/mounts`)
{
    ($fsname, $fstype) = m/\S+ (\S+) (\S+)/;
    $fsname =~ s!/!\\/!g;

    print "\t,\n" if not $first;
    $first = 0;
```

```
    print "\t{\n";
    print "\t\t\"{#FSNAME}\":\"$fsname\",\n";
    print "\t\t\"{#FSTYPE}\":\"$fstype\"\n";
    print "\t}\n";
}

print "\n\t]\n";
print "}\n";
```

Allowed symbols for LLD macro names are **0-9** , **A-Z** , **_** , **.**

Lowercase letters are not supported in the names.

An example of its output (reformatted for clarity) is shown below. JSON for custom discovery checks has to follow the same format.

```
{
  "data":[
  { "{#FSNAME}":"\/",                                  "{#FSTYPE}":"rootfs"   },
  { "{#FSNAME}":"\/sys",                               "{#FSTYPE}":"sysfs"    },
  { "{#FSNAME}":"\/proc",                              "{#FSTYPE}":"proc"     },
  { "{#FSNAME}":"\/dev",                               "{#FSTYPE}":"devtmpfs" },
  { "{#FSNAME}":"\/dev\/pts",                          "{#FSTYPE}":"devpts"   },
  { "{#FSNAME}":"\/",                                  "{#FSTYPE}":"ext3"     },
  { "{#FSNAME}":"\/lib\/init\/rw",                     "{#FSTYPE}":"tmpfs"    },
  { "{#FSNAME}":"\/dev\/shm",                          "{#FSTYPE}":"tmpfs"    },
  { "{#FSNAME}":"\/home",                              "{#FSTYPE}":"ext3"     },
  { "{#FSNAME}":"\/tmp",                               "{#FSTYPE}":"ext3"     },
  { "{#FSNAME}":"\/usr",                               "{#FSTYPE}":"ext3"     },
  { "{#FSNAME}":"\/var",                               "{#FSTYPE}":"ext3"     },
  { "{#FSNAME}":"\/sys\/fs\/fuse\/connections", "{#FSTYPE}":"fusectl"  }
  ]
}
```

Then, in the discovery rule's “Filter” field, we could specify “{#FSTYPE}” as a macro and “rootfs|ext3” as a regular expression.

You don't have to use macro names FSNAME/FSTYPE with custom LLD rules, you are free to use whatever names you like.