3 自动发现□LLD□

概述

自动发现□LLD□提供了一种在计算机上为不同实体自动创建监控项，触发器和图形的方法。例如□Zabbix可以在你的机器上自动开始监控文件系统或网络接口，而无需为每个文件系统或网络接口手动创建监控项。此外，可以配置Zabbix根据定期执行发现后的得到实际结果，来移除不需要的监控。

用户可以自己定义发现类型，只要它们遵循特定的JSON协议。

发现过程的一般架构如下。

首先，用户在“配置”→“模板”→“发现”列中创建一个发现规则。发现规则包括（1）发现必要实体（例如，文件系统或网络接口）的项 和（2）应该根据该项的值创建的监控项，触发器和图形的原型

发现所需实体的项就像其他地方所看到的常规项一样：服务器□server□向Zabbix agent□或者对应该项的其他类型的设置）查询该项的值□agent以文本值进行响应。区别在于agent响应的值应该包含特定JSON格式的已发现实体的列表。虽然这个列表的详细信息仅对自定义发现检查来说很重要，但有必要知道返回的值包含宏 - >值对的列表。 例如，项目“net.if.discovery”可能会返回两对□“{#IFNAME}” - >“lo”和 “{#IFNAME}” - >“eth0”□

这些宏用于名称，键值和其他原型字段中，然后用接收到的值为每个发现的实体创建实际的监控项，触发器，图形甚至主机。请参阅使用LLD宏选项的完整列表。 。

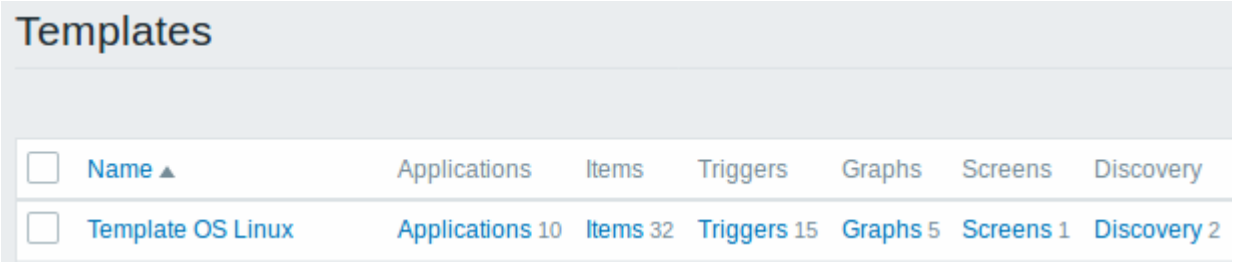
当服务器接收到已发现项的值时，它会查看宏→值对，每对都根据原型生成实际监控项，触发器和图形。在上面的“net.if.discovery”示例中，服务器将生成环路接口“lo”的一组监控项，触发器和图表，另一组用于界面“eth0”□

配置低级别发现□LLD□

我们来用文件系统发现为例子说明低级别发现□LLD□□

请执行以下操作，配置发现：

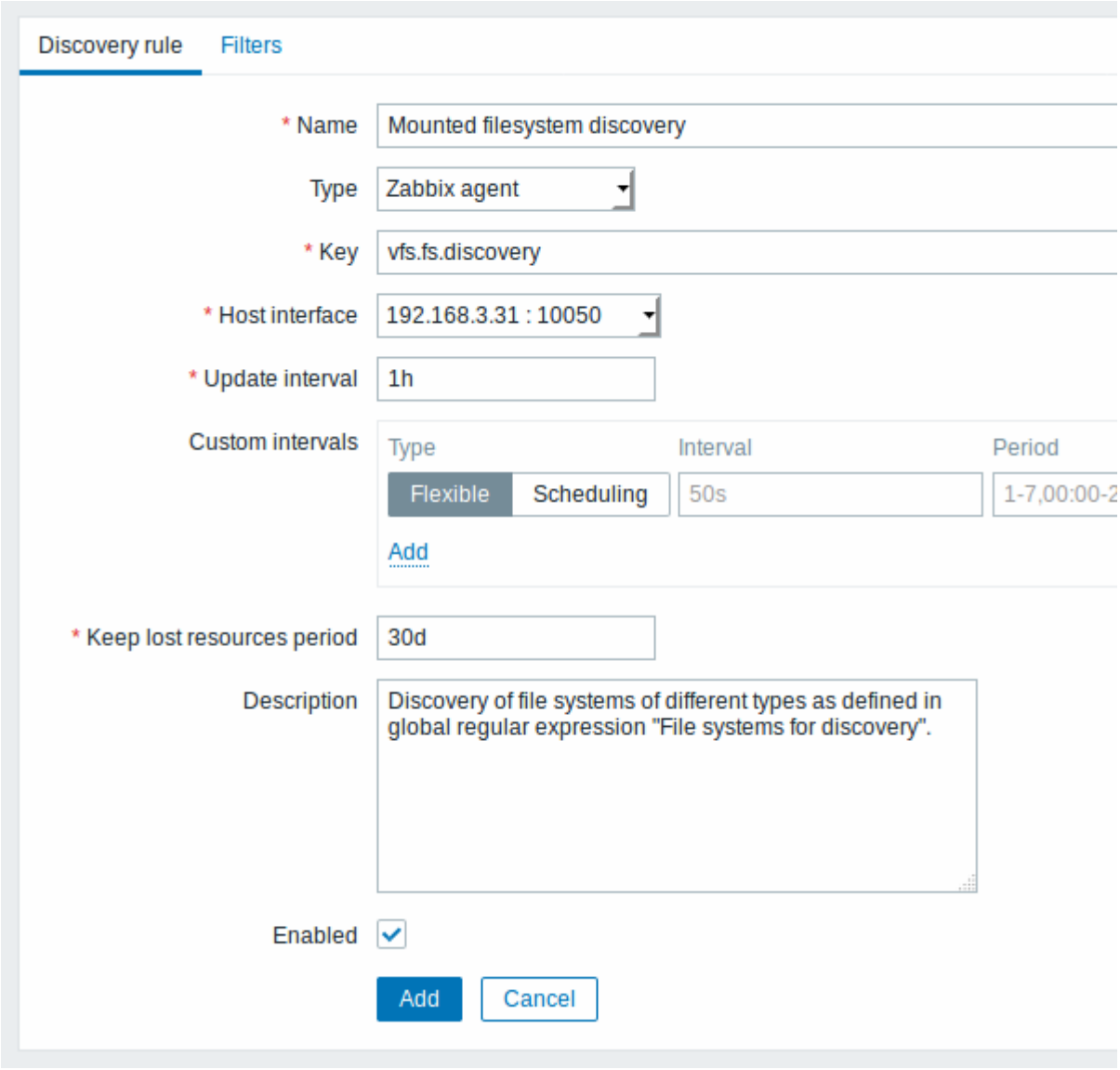
- 进入：*配置* → *模板*
- 选择一个合适的模板的行点击*发现*



- 单击屏幕右上角的 *创建发现规则*
- 填写需要的详细信息

发现规则

发现规则 选项卡包含常规发现规则属性：



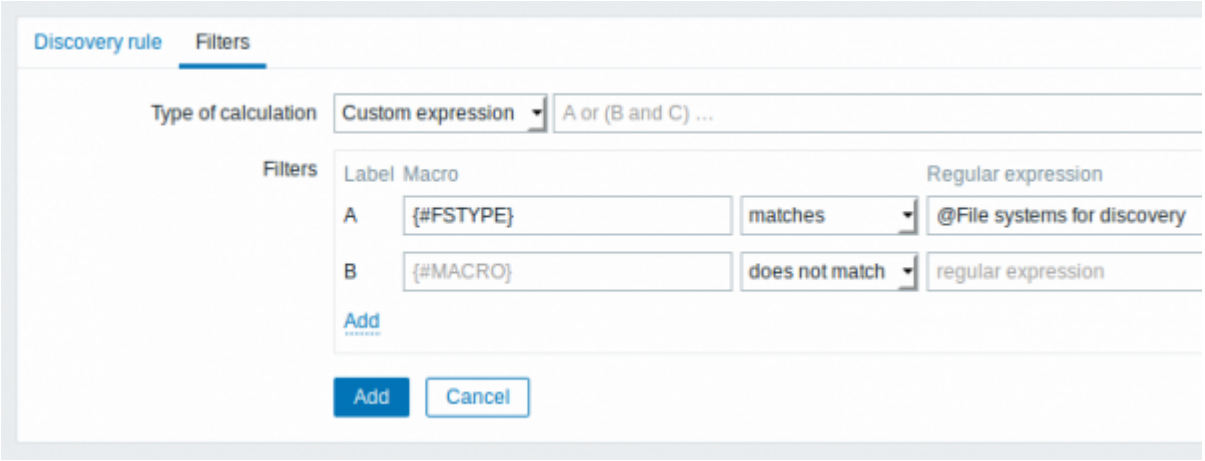
所有必填输入字段都标有红色星号。

参数	描述
名称	发现规则名称。
类型	执行发现的检查类型：可以是 *Zabbix agent*或*Zabbix agent*（*主动*）文件系统发现。
键值	自2.0版以来，许多平台上Zabbix agent中都内置了一个带有"vfs.fs.discovery"键的项目，(详情参见支持项目键列表)，将返回一个JSON□ 包含计算机上存在的文件系统列表及其类型详情。
数据更新间隔(秒)	此字段指定Zabbix执行发现的频率。 在开始刚刚设置文件系统发现时，可以将其设置为一个小间隔，但是一旦知道它可以工作之后，可以将其设置为30分钟或更长时间，因为文件系统通常不会经常更改。 从Zabbix 3.4.0起支持时间后缀 ，例如 30s□1m□2h□1d□ 自Zabbix 3.4.0起支持用户宏□ *注意*：如果设置为'0'，则不会轮询该项目。 但是，如果灵活间隔也存在非零值，则在灵活间隔持续时间内将轮询该项目。 *注意*对于现有发现规则，可以通过点击 *立即检查*按钮立即执行发现。
隔	您可以创建用于检查项目的自定义规则： **灵活** - 创建例外的 *更新间隔*（不同频次的间隔） **调度** - 创建自定义轮询调度。 有关详细信息，请参阅自定义时间间隔. 从Zabix 3.0.0起支持调度。
丢失资源保留期(天)	
Description	输入一段描述
Enabled	如果选中，规则将会处理该规则。

发现规则过滤器

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

过滤器 选项卡包括发现规则过滤器定义：



Parameter	Description
参数	**描述**
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** - offers the possibility to define a custom calculation of filters. The formula must include all filters in the list. Limited to 255 symbols.
Filters	A filter can be used to generate real items, triggers, and graphs only for certain file systems. It expects a Perl Compatible Regular Expression (PCRE). 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") and by drive types (supported only by Windows agent) using {#FSDRIVETYPE} macro (e.g., "fixed"). 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` {#FSDRIVETYPE} macro on Windows is supported since Zabbix **3.0.0**. 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. Filter drop-down offers two values to specify whether a macro matches a regular expression or does not match.
计算类型	可以使用以下计算过滤器的选项： **And** - 所有过滤器必须通过； **Or** - 只需要一个过滤器通过就足够了； **And/Or** - 使用具有不同的宏名称的 *And* ， 具有相同宏名称的*Or*； **Custom expression** - 提供定义过滤器的自定义计算的可能性。 公式必须包含列表中的所有过滤器。 限255个符号。

Parameter	Description
参数	**描述**
过滤器	过滤器可以用来生成真实监控项，触发器，但是图表仅用于特定的文件系统。它期待一个 Perl Compatible Regular Expression (PCRE)。例如，如果你只对C:, D:, 和 E: 文件系统有想法，你可以把 {#FSNAME} 放进 “宏” 中 并将 "^C|^D|^E" 正则表达式放入 “正则表达式” 文本字段。使用{#FSTYPE}宏（例如 "^ext|^reiserfs"）的文件系统类型和使用{#FSDRIVETYPE}宏（例如“fixed”）的驱动器类型（仅由Windows agent支持）也可以进行过滤。 您可以在“正则表达式”字段中输入正则表达式或引用全局 正则表达式 你可以用“grep -E”来测试一个正则表达式，例如： `for f in ext2 nfs reiserfs smbfs; do echo $f | grep -E '^ext|^reiserfs' || echo "SKIP: $f"; done` {#FSDRIVETYPE} 宏从Zabbix **3.0.0** 在Windows上开始支持。 从Zabbix **2.4.0** 开始支持定义多个过滤器。 注意，如果一些来自过滤器在响应中丢失，那发现的实体将会被忽略。\\“过滤器”下拉列表提供两个值，用于指定宏是与正则表达式匹配还是不匹配。

如果要正确发现仅按大小写不同的文件系统名称，则必须将MySQL中的Zabbix数据库创建为区分大小写。Zabbix database in MySQL must be created as case-sensitive if file system names that differ only by case are to be discovered correctly.
在正则表达式中的错误或错字，应用在LLD规则中时可能会导致删除删除数以千计的配置元素，历史值和许多主机的活动。例如，错误的“用于发现的文件系统”正则表达式可能会导致删除数以千计的监控项，触发器，历史值和活动。The mistake or typo in regex used in LLD rule may cause deleting thousands of configuration elements, historical values and events for many hosts. For example, incorrect “File systems for discovery” regular expression may cause deleting thousands of items, triggers, historical values and events.
Discovery rule history is not preserved.
不保留发现规则历史记录。

Form buttons

表格按钮

在表格底部的按钮会显示许多可用的操作。

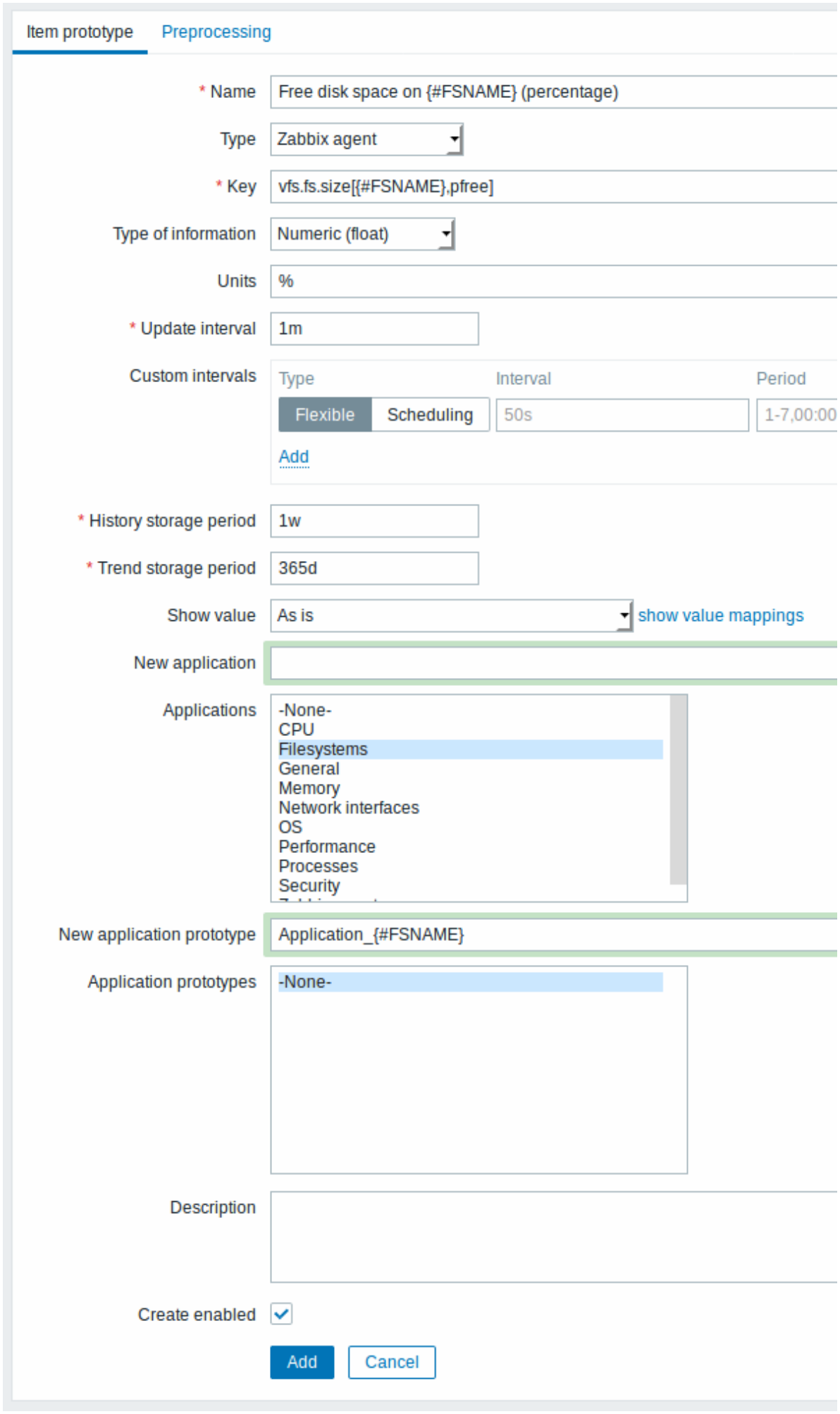
Add	Add a discovery rule. This button is only available for new discovery rules.
Update	Update the properties of a discovery rule. This button is only available for existing discovery rules.
Clone	Create another discovery rule based on the properties of the current discovery rule.
Check now	Perform discovery based on the discovery rule immediately. The discovery rule must already exist. See more details. *Note* that when performing discovery immediately, configuration cache is not updated, thus the result will not reflect very recent changes to discovery rule configuration.
Delete	Delete the discovery rule.
Cancel	Cancel the editing of discovery rule properties.
Add	新增一个发现规则，此按钮仅可用于新的发现规则。
Update	更新发现规则的属性。 此按钮仅适用于现有发现规则。
Clone	在现有的发现规则的属性基础上创建另一个发现规则。

Check now	立即根据发现规则执行发现。 发现规则必须已存在。 查看 详情. *注意* 当立即执行发现时，配置缓存不会更新，因此结果并不代表发现规则配置中最新的改变。
Delete	删除发现规则。
Cancel	取消编辑发现规则属性。

Item prototypes

监控项原型

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. 一旦规则创建完成了，找到那条规则下的监控项，并点击“创建原型”来创建一个监控项原型。请注意在需要文件系统名称的情况下如何使用宏 {#FSNAME} □ 处理发现规则时，此宏将替换为发现的文件系统。


Item prototype
Preprocessing
* Name
Free disk space on {#FSNAME} (percentage)
Type
Zabbix agent
* Key
vfs.fs.size[{#FSNAME},pfree]
Type of information
Numeric (float)
Units
%
* Update interval
1m
Custom intervals
Type
Interval
Period
Flexible
Scheduling
50s
1-7,00:00
Add
* History storage period
1w
* Trend storage period
365d
Show value
As is
show value mappings
New application
Applications
-None-
CPU
Filesystems
General
Memory
Network interfaces
OS
Performance
Processes
Security
New application prototype
Application_{#FSNAME}
Application prototypes
-None-
Description
Create enabled
Add
Cancel

Low-level discovery macros and user macros may be used in item prototype configuration and item value preprocessing parameters.

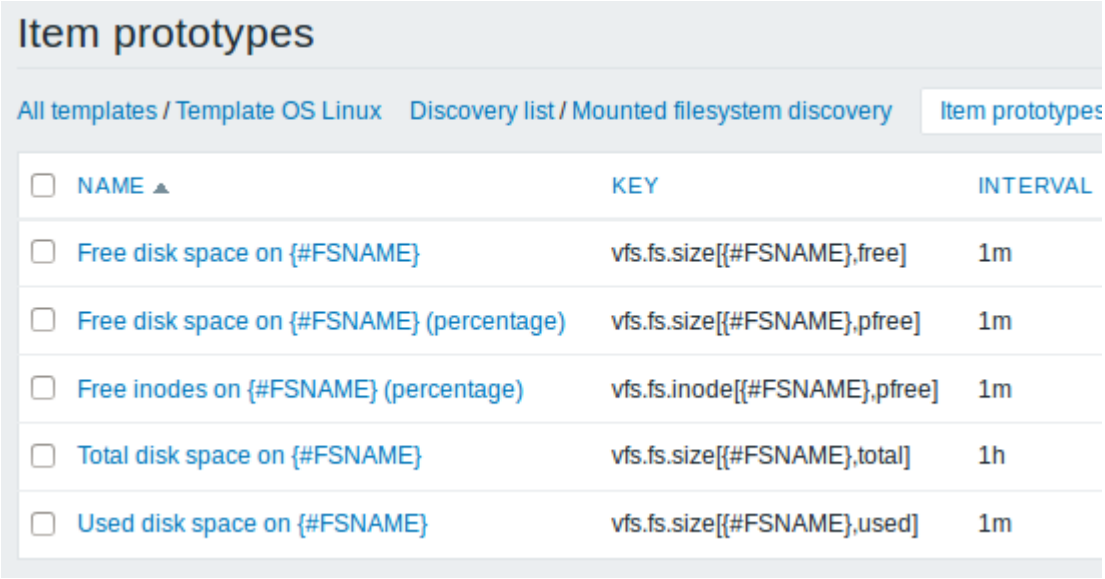
低级别发现 宏 and 用户 宏 可能会被用在监控项原型配置和监控项值预处理 参数.

Context-specific escaping of low-level discovery macros is performed for safe use in regular expression and XPath preprocessing parameters.执行特定于上下文的低级别发现宏的转义，以便在正则表达式和XPath预处理参数中安全使用。

Attributes that are specific for item prototypes: 特定于监控项原型的属性：

参数	描述
New application prototype	You may define a new application prototype. In application prototypes you can use low-level discovery macros that, after discovery, will be substituted with real values to create applications that are specific for the discovered entity. See also application discovery notes for more specific information.
Application prototypes	Select from the existing application prototypes.
Create enabled	If checked the item will be added in an enabled state. If unchecked, the item will be added to a discovered entity, but in a disabled state.
全新的应用原型	你可能定义一个新的应用原型。 在应用原型中，你可以使用低级别发现宏，在发现后，将替换为实际值以创建特定于已发现实体的应用程序.在 应用发现注意事项 查看更详细的信息。
应用程序原型	从现有的应用程序原型选择。
创建已启用	如果选中，则监控项将以启用状态添加。 如果未选中，则该监控项将添加到已发现的实体，但处于禁用状态。

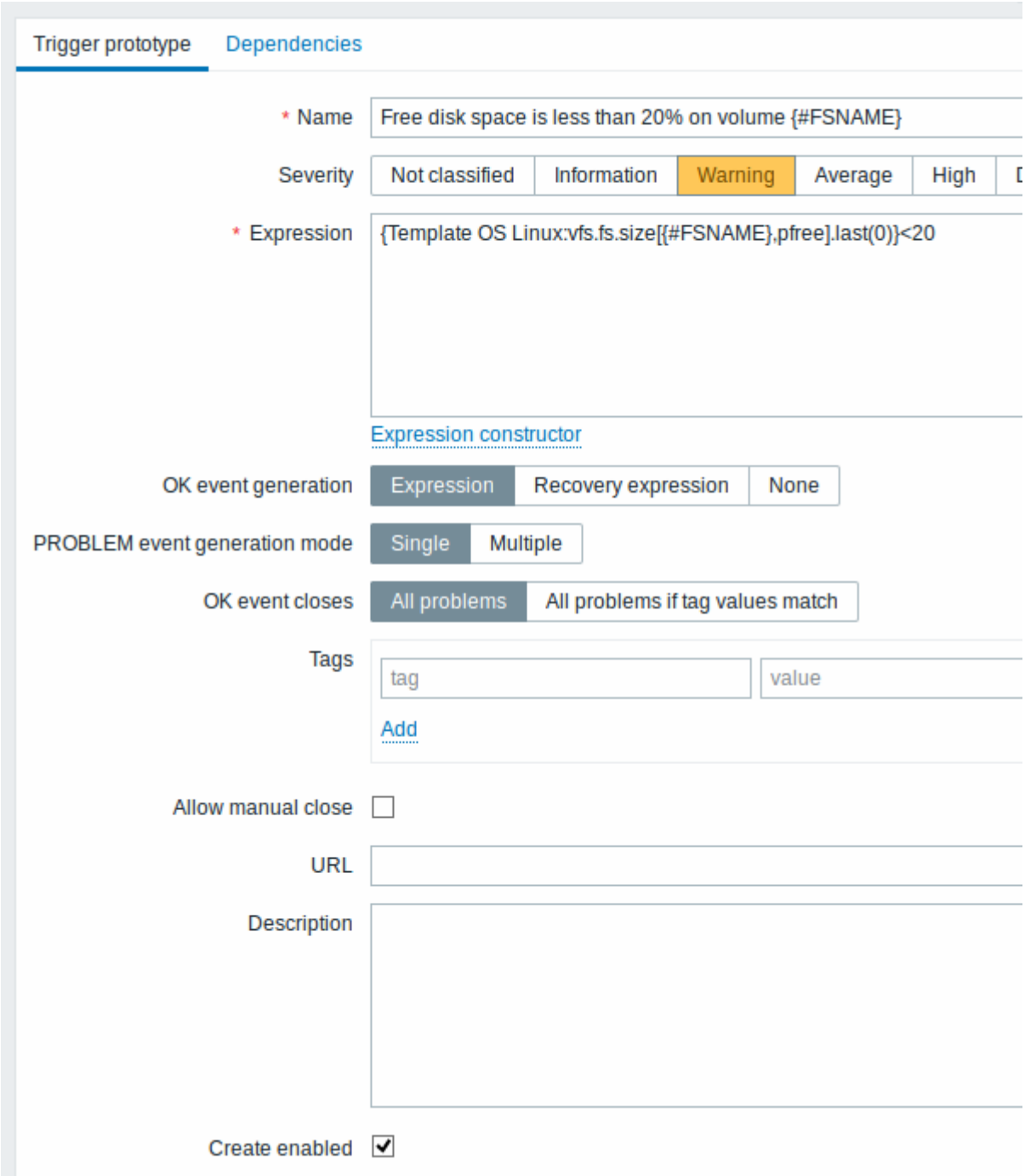
We can create several item prototypes for each file system metric we are interested in: 我们可以根据需求为每个文件系统指标创建许多监控项原型：



Trigger prototypes

触发器原型

We create trigger prototypes in a similar way as item prototypes: 我们创建触发器原型的方式和创建监控项原型的方式相似：



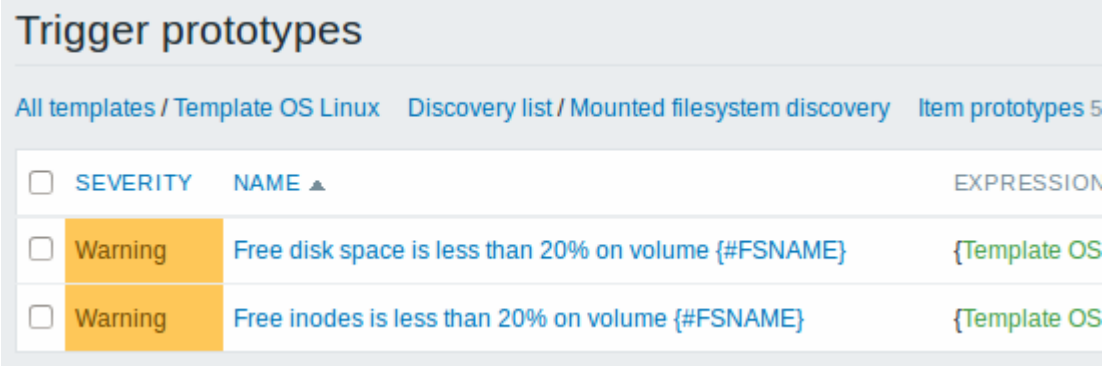
Attributes that are specific for trigger prototypes: 特定于触发器原型的属性：

参数	描述
Create enabled	If checked the trigger will be added in an enabled state. If unchecked, the trigger will be added to a discovered entity, but in a disabled state.
创建已启用	如果选中，则触发器将以启用状态添加。 如果未选中，则该触发器将添加到已发现的实体，但处于禁用状态。

When real triggers are created from the prototypes, there may be a need to be flexible as to what constant ('20' in our example) is used for comparison in the expression. See how user macros with context can be useful to accomplish such flexibility.

当真实的触发器从原型创建，可能需要灵活地确定在表达式中用于比较的常量（在我们的示例中为'20'）。了解 基于环境的用户宏 如何有助于实现这种灵活性。

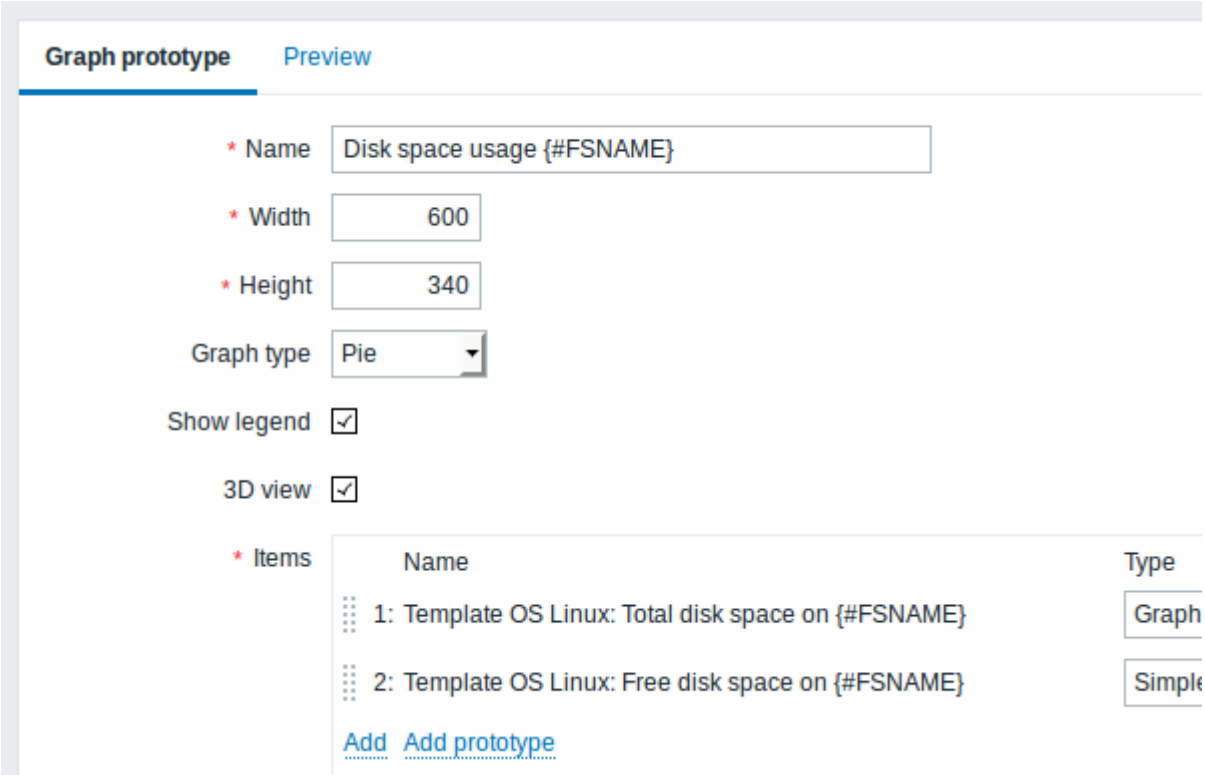
You can define dependencies between trigger prototypes as well (supported since Zabbix 3.0). To do that, go to the Dependencies tab. A trigger prototype may depend on another trigger prototype from the same low-level discovery (LLD) rule or on a regular trigger. A trigger prototype may not depend on a trigger prototype from a different LLD rule or on a trigger created from trigger prototype. Host trigger prototype cannot depend on a trigger from a template. 您也可以定义触发器原型之间的（从Zabbix 3.0开始支持□□You can define 依赖关系（从Zabbix 3.0开始支持）。为此，转到 *依赖关系* 选项卡。 触发器原型可能依赖于来自相同低级别发现□LLD□规则的另一个触发器原型或常规触发器。触发器原型可能不依赖于来自不同LLD规则的触发器原型或依赖于触发器原型创建的触发器。主机触发器原型不能依赖于来自模板的触发器。

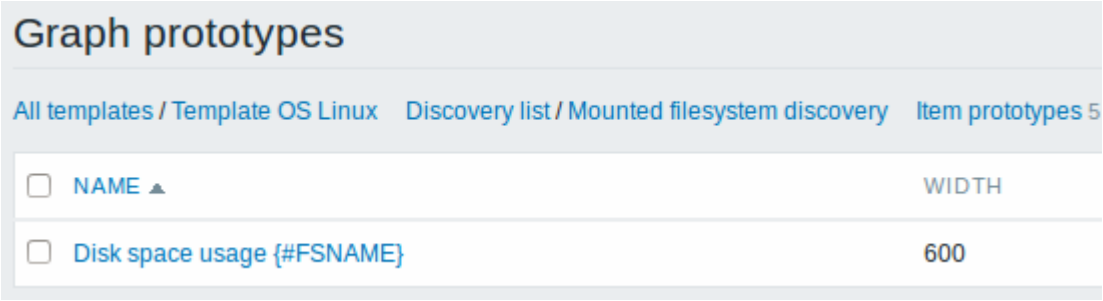


Graph prototypes

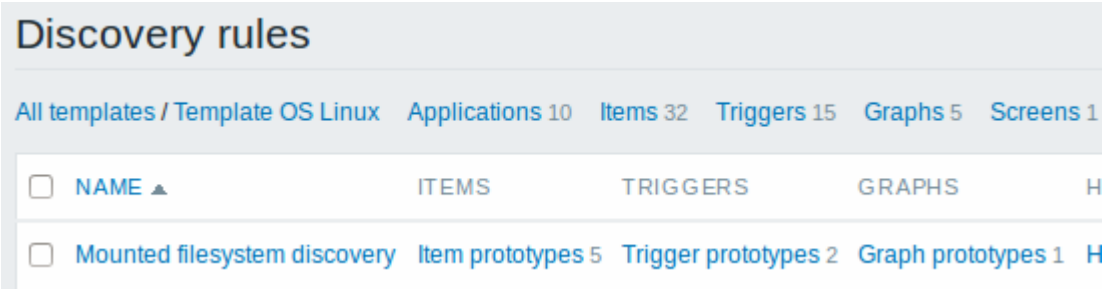
图表原型

We can create graph prototypes, too: 我们也能创建图表原型：





Finally, we have created a discovery rule that looks like shown below. It has five item prototypes, two trigger prototypes, and one graph prototype. 最后，我们像下面的展示的一样创建一个发现规则。有五个监控项原型，两个触发器原型，和一个图表原型。



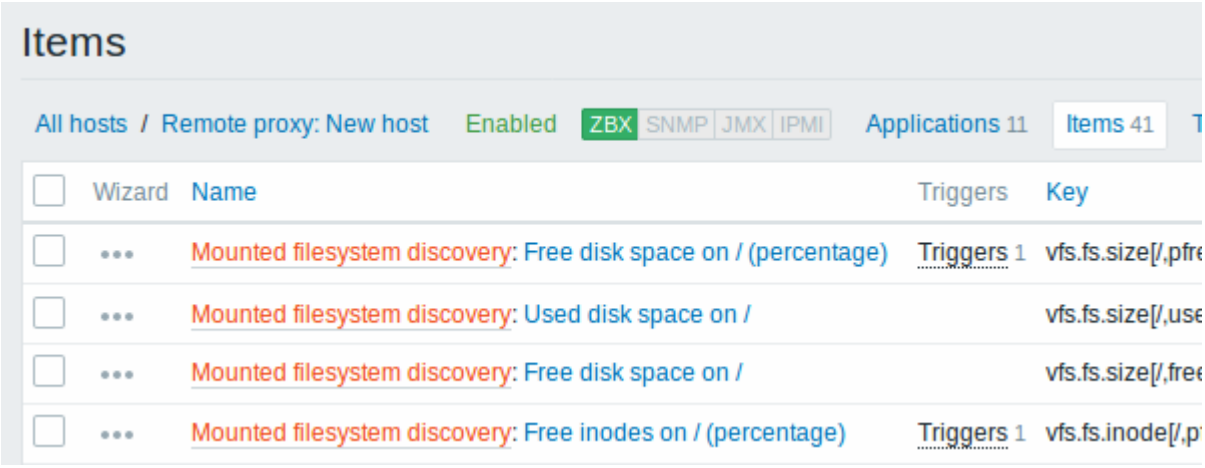
注意: For configuring host prototypes, see the section about host prototype configuration in virtual machine monitoring. *注意*：有关配置主机原型的信息，请参阅虚拟机监视中有关主机原型配置的部分。

Discovered entities

发现的实体

The screenshots below illustrate how discovered items, triggers, and graphs look like in the host's

configuration. Discovered entities are prefixed with an orange link to a discovery rule they come from. 下面的屏幕截图说明了主机配置中发现的项目，触发器和图形的外观。 发现的实体的前缀是橙色链接，指向它们来自的发现规则。

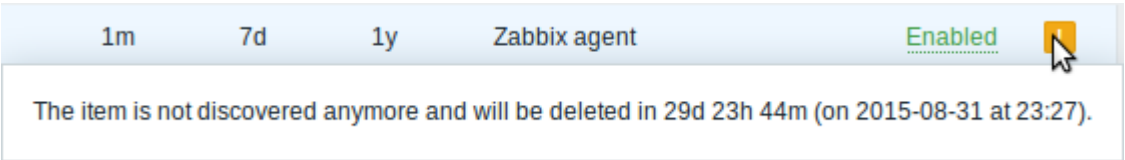


Note that discovered entities will not be created in case there are already existing entities with the same uniqueness criteria, for example, an item with the same key or graph with the same name. 请注意，如果已存在具有相同唯一性条件的实体，则不会创建已发现的实体，例如，具有相同key或具有相同名称的图形的监控项。

Items (similarly, triggers and graphs) created by a low-level discovery rule 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. 如果一个发现的实体（文件系统，接口等）停止被发现（或是再也不通过过滤器），使用低级别发现规则创建的监控项（触发器和图表也相似）将会自动被删除。在这种情况下，在Keep lost resources period字段传递中定义的日期之后，将删除监控项，触发器和图表。

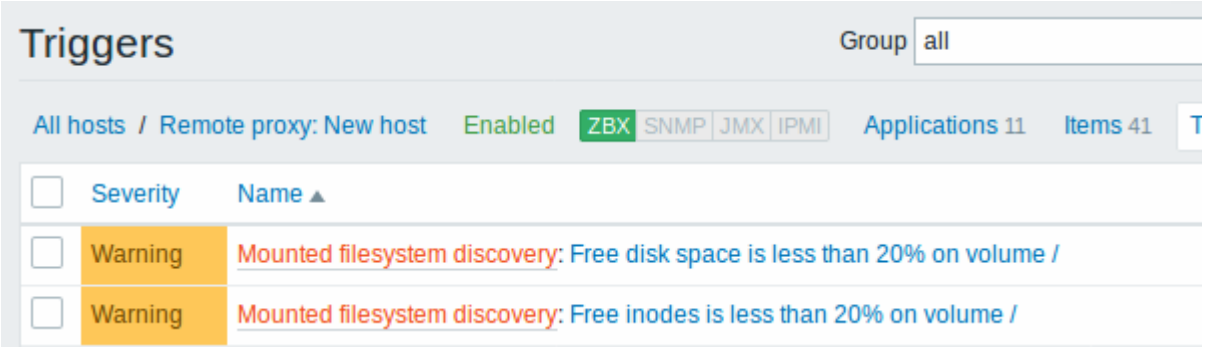
When discovered entities become 'Not discovered anymore', a lifetime indicator is displayed in the item list. Move your mouse pointer over it and a message will be displayed indicating how many days are left until the item is deleted.

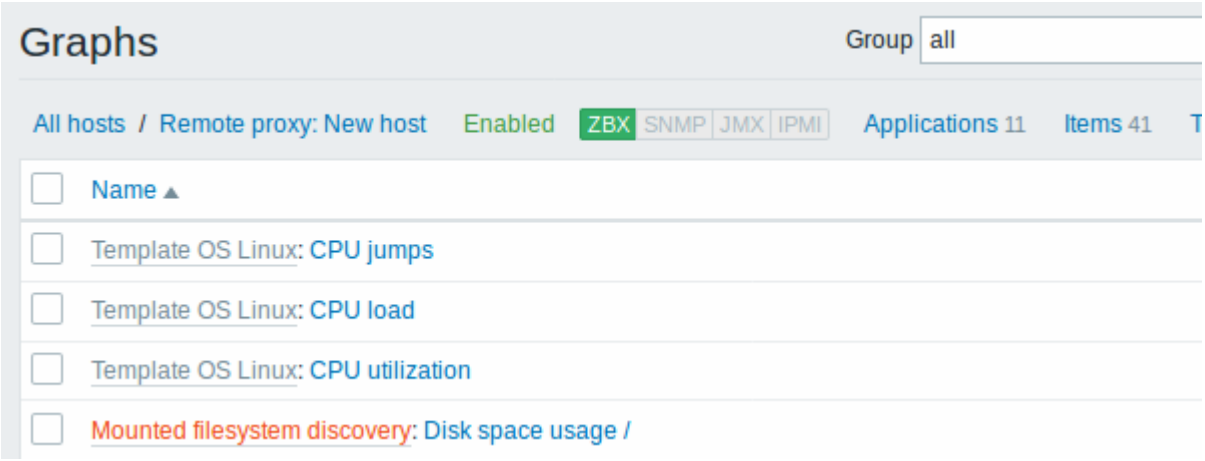
当发现实体转变成“不再发现”状态，生命周期指示符显示在监控项列表中。将鼠标指针移到它上面，将显示一条消息，表示在删除项目之前剩余的天数。



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. 如果实体已标记为删除但未在预期时间删除（已禁用的发现规则或项目主机），则下次处理发现规则时将删除这些实体。

Entities containing other entities, which are marked for deletion, will not update if changed on the discovery rule level. For example, LLD-based triggers will not update if they contain items that are marked for deletion. 如果在发现规则级别更改，则包含标记为删除的其他实体的实体将不会更新。 例如，如果基于LLD的触发器包含标记为删除的监控项，则不会更新。





Other types of discovery

其他类型的发现

More detail and how-tos on other types of out-of-the-box discovery is available in the following sections:

以下部分提供了有关其他类型的开箱即用发现的更多详细信息和方法：

- discovery of network interfaces;
- discovery of CPUs and CPU cores;
- discovery of SNMP OIDs;
- discovery of JMX objects;
- discovery using ODBC SQL queries;
- discovery of Windows services;
- discovery of host interfaces in Zabbix.
- 网络接口发现；
- CPUs 和 CPU 核心发现；
- SNMP OIDs发现；
- JMX 对象发现；
- 使用ODBC SQL 查询发现；
- Windows services发现；
- 在Zabbix中的主机接口发现

For more detail on the JSON format for discovery items and an example of how to implement your own file system discoverer as a Perl script, see creating custom LLD rules. 有关发现项的JSON格式的更多详细信息以及如何将自己的文件系统发现者实现为Perl脚本的示例，请参阅创建自定义LLD规则

Data limits for return values

反馈值的数据限制

There is no limit for low-level discovery rule JSON data if it is received directly by Zabbix server, because return values are processed without being stored in a database. There's also no limit for custom low-level discovery rules, however, if it is intended to acquire custom LLD data using a user parameter, then user parameter return value limit applies (512 KB). 如果是直接来自Zabbix server那对低级别发现规则JSON数据没有限制，因为反馈值是在没有存储在数据库中的情况下处理的。对于定制的低级别发现规则也没有限制，但是，如果 如果要使用用户参数获取自定义LLD数据，则应用用户参数反馈值会有限制512 KB

If data has to go through Zabbix proxy it has to store this data in database so database limits apply, for example, 2048 bytes on a Zabbix proxy run with IBM DB2 database.

如果数据必须通过Zabbix proxy则必须将此数据存储在数据库中，以便应用数据库限制，例如，在运行IBM DB2数据库的Zabbix proxy上应用2048字节。

Multiple LLD rules for same item

同一监控项的多个LLD规则

Since Zabbix agent version 3.2 it is possible to define several low-level discovery rules with the same discovery item.

自从Zabbix agent 3.2版本开始，就可以使用相同的发现监控项定义许多低级别发现规则了。

To do that you need to define the Alias agent parameter, allowing to use altered discovery item keys in different discovery rules, for example `vfs.fs.discovery[foo]`, `vfs.fs.discovery[bar]`, etc. 为此，你需要定义Alias agent 参数，在不同的发现规则中允许使用更改发现监控项密钥。例如，`vfs.fs.discovery[foo]`, `vfs.fs.discovery[bar]`等。

Creating custom LLD rules

创建自定义LLD规则

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

也可以创建一个完整的自定义LLD规则，同时发现任何类型的实体——例如，在database server上的数据库。

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.

为此，应创建一个反馈JSON的自定义项，指定找到的对象并可选——他们的一些属性。 每个实体宏不受限制——虽然内置的发现规则反馈一个或两个宏（例如，两个用于文件系统发现），单反馈更多宏也是可能的。

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”:

这里有个例子可以最好地证明需要的JSON格式。假设我们正在运行一个接的Zabbix 1.8 agent不支持“vfs.fs.discovery”但我们仍旧需要发现文件系统。这有一个简单的Linux Perl脚本，可以发现挂载的文件系统，并输出JSON其中包括文件系统的名称和类型。一种使用它的方式是使用键“vfs.fs.discovery_perl”作为UserParameter

```
#!/usr/bin/perl

$first = 1;

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

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

    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.

LLD宏名称的允许符号为 **0-9** , **A-Z** , **_** , **.**

名称中不支持小写字母。

其输出示例（为清晰起见重新格式化）如下所示。 自定义发现检查的JSON必须遵循相同的格式。

```
{
  "data":[
  { "{#FSNAME}":"/",                              "{#FSTYPE}":"rootfs"   },
  { "{#FSNAME}":"/sys",                           "{#FSTYPE}":"sysfs"    },
  { "{#FSNAME}":"/proc",                          "{#FSTYPE}":"proc"     },
  { "{#FSNAME}":"/dev",                           "{#FSTYPE}":"devtmpfs" },
  { "{#FSNAME}":"/dev/pts",                       "{#FSTYPE}":"devpts"   },
  { "{#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.

然后，在发现规则中的“过滤器”部分，我们可以指定“{#FSTYPE}“作为一个宏“rootfs|ext3” 作为一个正则表达式。

You don't have to use macro names FSNAME/FSTYPE with custom LLD rules, you are free to use whatever names you like.
你不必要使用自定义LLD规则的宏名称 FSNAME/FSTYPE 你可以使用任何你喜欢的名称。

Note that, if using a user parameter, the return value is limited to 512 KB. For more details, see data limits for LLD return values.

需要注意的是，如果使用一个用户参数，反馈值限制到512 KB更多信息，请参考LLD反馈值的数据限制.

Using LLD macros in user macro contexts

在用户宏环境中使用LLD宏

User macros with context can be used to accomplish more flexible thresholds in trigger expressions. Different thresholds may be defined on user macro level and then used in trigger constants depending on the discovered context. Discovered context appears when the low-level discovery macros used in the macros are resolved to real values.

环境中的用户宏可用于在触发器表达式中实现更灵活的阈值。不同的阈值 可以在用户宏级别上定义不同的阈值，然后根据发现的环境将其用于触发器常量。当宏中使用的 低级别发现宏 被解析为实际值时，将显示已发现的环境。

To illustrate we can use data from the example above and assume that the following file systems will be discovered: `/`, `/home`, `/tmp`, `/usr`, `/var`.

We may define a free-disk-space trigger prototype for a host, where the threshold is expressed by a user macro with context:

为了证明我们可以使用上述的例子中的数据， 假设将发现以下文件系统： /, /home, /tmp, /usr, /var.

我们可以为主机定义一个自由磁盘空间触发器原型，其中阈值由具有发现环境的用户宏表示：

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

Then add user macros: 然后新增用户宏:

- {$LOW_SPACE_LIMIT} **10**
- {$LOW_SPACE_LIMIT:/home} **20**
- {$LOW_SPACE_LIMIT:/tmp} **50**

Now, once the file systems are discovered, events will be generated if /, /usr and /var filesystems have less than **10**% of free disk space, the /home filesystem - less than **20**% of free disk space or the /tmp filesystem - less than **50**% of free disk space.

现在，一旦文件系统被发现了，如果/, /usr and /var文件系统有少于**10**%的自由磁盘空间，/home文件系统——少于**20**%的自由磁盘空间或/tmp文件系统——少于**50**%的自由磁盘空间，将生成事件。