1 Monitoring Windows

So far, we have explored the monitoring of various services and Linux systems. While monitoring Microsoft Windows is very similar in many aspects, some Windows-specific support is available in Zabbix. Most of the things we learned about the Zabbix agent, using items and even user parameters, are still relevant on Windows. In this chapter, we will explore installing the Zabbix agent on Windows, monitoring Windows performance counters, and using the built-in **Windows Management Instrumentation** (**WMI**) support. We will also try out Windows service monitoring, including the ability to discover them automatically, and the event log system support in the Zabbix agent. For this section, you will need a Windows machine that is accessible from the Zabbix server.

In this chapter, we will cover the following topics:

- Installing the Zabbix agent for Windows
- Querying performance counters
- Querying WMI
- Monitoring Windows services
- Windows event-log monitoring

Installing the Zabbix agent for Windows

To install the agent, it first has to be obtained. On Windows, compiling software is less common and most users get binary distributions, which is exactly what we will do now.

The Windows build of the Zabbix agent can be obtained from two official locations—either from the download page at `http://www.zabbix.com/download.php`, or from the source archive. While the practice of keeping binaries in the sources is not suggested, that's how Zabbix does it and sometimes, we can use it to our advantage. If you installed from source, it might be a good idea to use the Windows agent binary from the same archive so that the versions match. The agent executable is located in the `bin/win32` or `bin/win64` subdirectory—choose the one that is appropriate for your architecture. If you installed from the packages, visit the download page and grab the Windows agent archive, but make sure to use the same or older major version of the Zabbix server. With the agent at hand one way or another, place it in the same directory on the Windows machine.

For simplicity, we'll use `C:\zabbix` this time, but you are free to use any other directory. We will also need the configuration file, so grab the example provided at `conf/zabbix_agentd.win.conf` if you used the binary from the sources, or from the `conf/` directory inside the archive if you downloaded the binaries from the Zabbix website. Place the configuration file in the same directory—there should be two files now. Before we continue with the agent itself, let's figure out whether we need to alter the configuration in any way. Open `C:\zabbix\zabbix_agentd.win.conf` in your favorite text editor and look for any parameters we might want to change. First, the log file location isn't quite right—it's set to `C:\zabbix_agentd.log`, so let's change it, as follows:

```
LogFile=c:\zabbix\zabbix_agentd.log
```

You can use both forward and back slashes on the Windows Zabbix agent daemon command line and in the configuration file.

We have already learned that the server line, which currently reads `Server=127.0.0.1`, will have to be changed. Replace the `127.0.0.1` part with the IP address of your Zabbix server. And to be sure that active items will work as expected, let's check the `Hostname` directive; it is set to the Windows host by default and we could leave it like that. Another parameter for active checks was `ServerActive`. Replace `127.0.0.1` here with the Zabbix server IP address as well and save the file.

If we were to start our agent now, it would automatically register on the Zabbix server, based on the configuration we created in Chapter 11, A*utomating Configuration*.

While it would be convenient, we want to test things in a stricter fashion this time—go to **Configuration | Actions**, switch to **Auto registration** in the **Event source** drop-down, and click on **Enabled**, next to **Testing registration**—this should disable the auto-registration we set up earlier.

Now let's try to start the agent up. Start the Windows `cmd.exe` and execute the following:

```
C:\zabbix>zabbix_agentd.exe -c c:/zabbix/zabbix_agentd.win.conf
```

You might have to prefix the commands with `.\` on some versions of Windows.
If you see no output or another window appears very briefly, you should start Command Prompt as the admin user. In recent versions of Windows, the menu entry is called **Command Prompt (Admin)**.

The agent daemon refuses to start up:

```
zabbix_agentd.exe [6348]: use foreground option to run Zabbix agent as
console application
```

Let's find out how we can supply the foreground option, then. The agent daemon executable on Windows has additional options that can be passed to it, so execute it in Command Prompt (when located in the directory where `zabbix_agentd.exe` resides):

```
C:\zabbix>zabbix_agentd.exe --help
```

Looking at the **Options** section, the foreground parameter is listed there:

```
-f --foreground   Run Zabbix agent in foreground
```

Let's try to use that option:

```
C:\zabbix>zabbix_agentd.exe --foreground -c
c:/zabbix/zabbix_agentd.win.conf
Starting Zabbix Agent [Windows host]. Zabbix 3.0.0 (revision 58455).
Press Ctrl+C to exit.
```

It looks like the agent has started up. For a quick test, try running the following from the Zabbix server. On the Zabbix server, execute the following:

```
$ zabbix_get -s <Windows host IP> -k system.cpu.load
0.316667
```

The agent is running and we can query values from it; it looks great. There's one issue, though—we are currently running it in our Terminal. If we were to close the Terminal, the agent wouldn't run anymore. If the system were rebooted, the agent would not be started automatically. Running the agent in the foreground is nice, but we can also run it as a Windows service. How, *exactly?* First, stop it by pressing *Ctrl* + *C*, then look at the `--help` output again. Among all the parameters, we are interested in the `Functions` section this time:

```
Functions:
-i --install       Install Zabbix agent as service
-d --uninstall     Uninstall Zabbix agent from service
-s --start         Start Zabbix agent service
-x --stop          Stop Zabbix agent service
```

`--multiple-agents` in the **Options** section is intended to run multiple agents on the same system as separate Windows services. If used, the service name will include the `Hostname` parameter value from the specified configuration file in the service name.

The Zabbix agent daemon for Windows includes the functionality to install it as a standard Windows service, which is controlled by the options in this section. Unless you are simply doing some testing, you'll want to properly install it, so let's do that now:

```
C:\zabbix>zabbix_agentd.exe -c c:/zabbix/zabbix_agentd.win.conf -i
```

A confirmation dialog might come up at this time. Click on **Yes**. If you were running Command Prompt as an administrative user, installing the service should succeed:

```
zabbix_agentd.exe [6248]: service [Zabbix Agent] installed
successfully
zabbix_agentd.exe [6248]: event source [Zabbix Agent] installed
successfully
```

If not, this command might fail with the following:

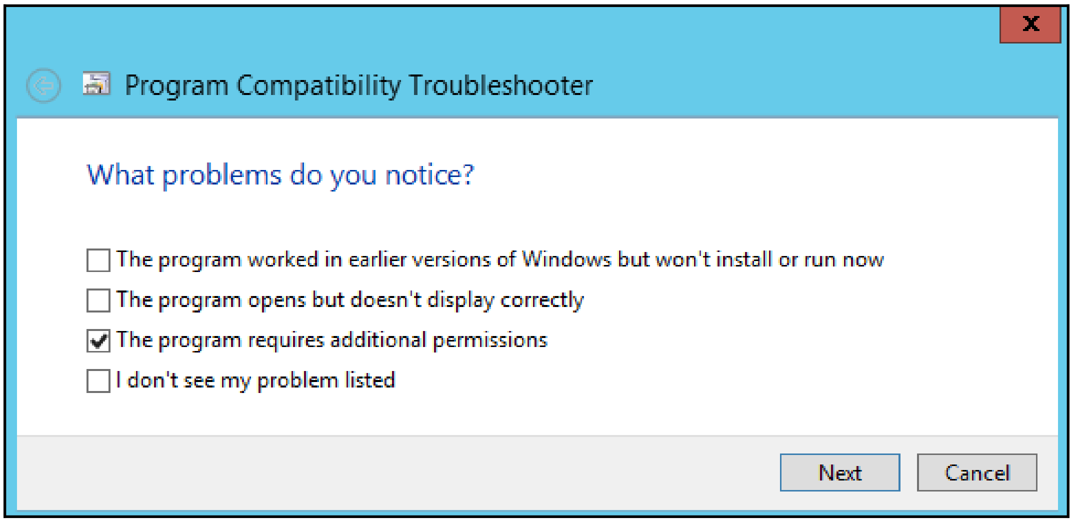
```
zabbix_agentd.exe [3464]: ERROR: cannot connect to Service Manager:
[0x00000005] Access is denied.
```

In this case, you should either run Command Prompt as an administrative user, or allow the program to run as the administrative user.

To do the latter, right-click on `zabbix_agentd.exe`, and choose **Troubleshoot compatibility**.

In the resulting window, perform the following steps:

1. Click on **Troubleshoot program** and mark the checkbox for **The program requires additional permissions**:



Program Compatibility Troubleshooter

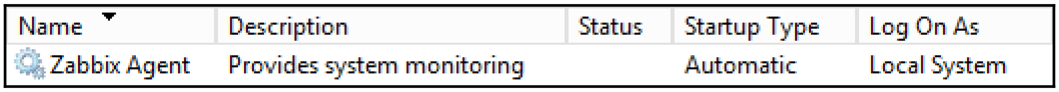
What problems do you notice?

- ☐ The program worked in earlier versions of Windows but won't install or run now
- ☐ The program opens but doesn't display correctly
- ☑ The program requires additional permissions
- ☐ I don't see my problem listed

2. Click on **Next**, then **Test the program**, and **Next** again
3. In the final window, choose **Yes**, save these settings for this program, then click on **Close**

If running the agent daemon seems to have no input or shows a window very briefly, use the administrative Command Prompt.

If everything was successful, the Zabbix agent daemon will have been installed as a Windows service using the configuration file, specified by the `-c` flag. You can verify, in the Windows **Control Panel** | **Services** section, that the Zabbix service has indeed been installed:

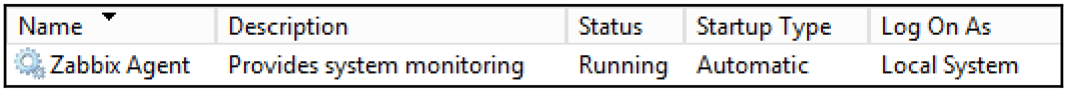


Name	Description	Status	Startup Type	Log On As
Zabbix Agent	Provides system monitoring		Automatic	Local System

While it has been set to start up automatically, it is stopped now. We can start it by either right-clicking on the **Zabbix Agent** service entry and choosing **Start**, or by using the command line to switch to `zabbix_agentd.exe`. Let's try the latter method now:

```
C:\zabbix>zabbix_agentd.exe --start
```

You might have to answer another security prompt here, but the service should start up successfully. We can verify in the services list that the Zabbix service has started up:



Name	Description	Status	Startup Type	Log On As
Zabbix Agent	Provides system monitoring	Running	Automatic	Local System

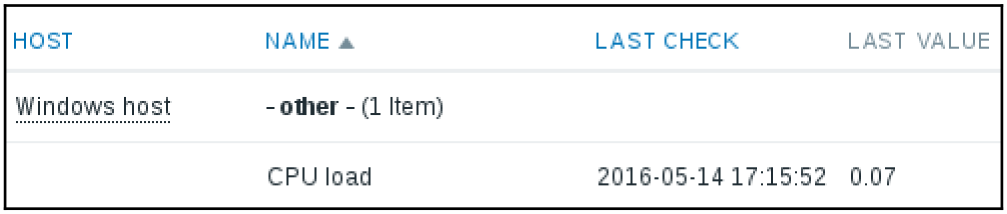
If you opened the service list earlier, refresh the contents by pressing *F5*.

It looks like everything is fine on the monitored host, which we will now have to configure in the frontend:

1. Open **Configuration** | **Hosts** and click on **Create host**, then fill in the following values:
    - **Host name**: `Windows host`
    - **Groups**: If there's any group in the **In groups** box, remove it
    - **New group**: `Windows servers`
    - **Agent interfaces, IP address**: Enter the IP address of that host

2. When done, click on the **Add** button at the bottom.
3. Now select **Windows servers** in the **Group** drop-down. Click on **Items** next to **Windows host**, then click on **Create item**. Enter these values:
    - **Name**: `CPU load`
    - **Key**: `system.cpu.load`
    - **Type of information**: **Numeric (float)**
4. When done, click on the **Add** button at the bottom.

We can now check out incoming data at **Monitoring** | **Latest data**—clear out the other filter fields, select **Windows host** in the **Host group** field, and then click on **Filter**:



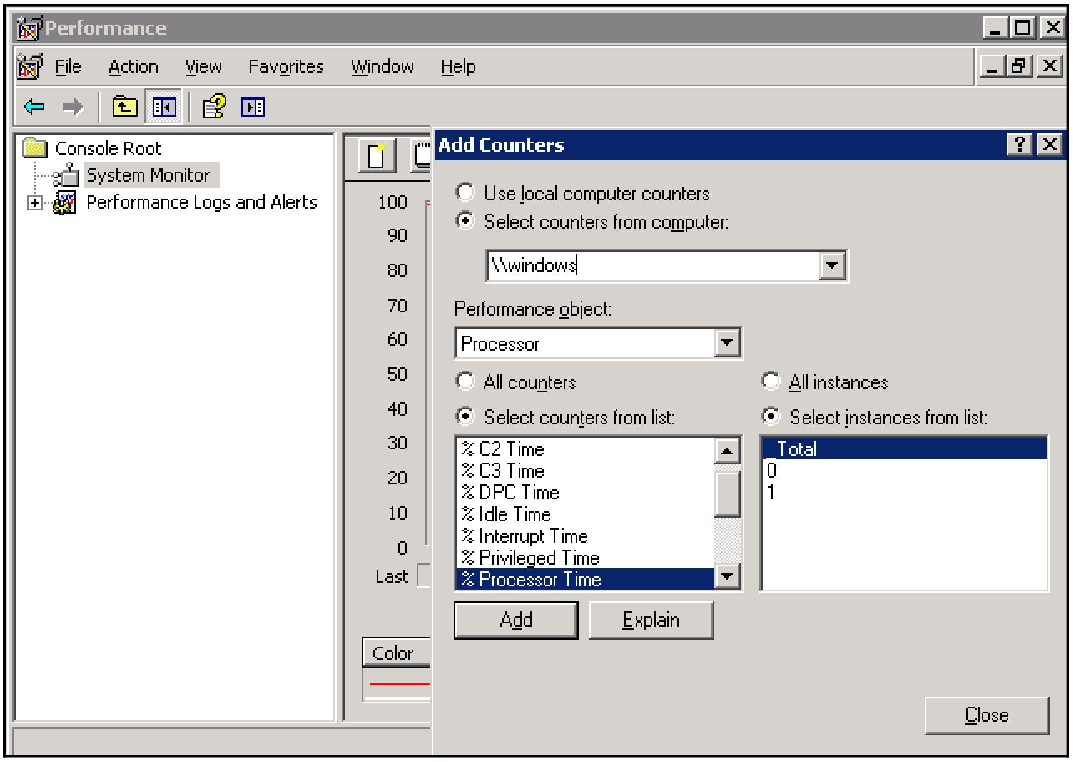
HOST	NAME ▲	LAST CHECK	LAST VALUE
Windows host	- **other** - (1 Item)		
	CPU load	2016-05-14 17:15:52	0.07

CPU load on Windows works in a similar manner as on Unix systems, although Windows administrators are less familiar with it. CPU utilization is more often used on Windows.

We have now successfully retrieved data on the CPU load for this Windows machine. Notice how the key syntax is the same as for Linux. This is true for several other keys, and you can check out the Zabbix documentation to determine which keys are supported on which platform.

Querying performance counters

While many keys match between platforms, there's a whole category that is specific to Windows. Zabbix supports Windows built-in metrics-gathering system—performance counters. People who are familiar with Windows probably know that these can be found at **Control Panel** | **Administrative Tools** | **Performance** in older versions of Windows, and **Administrative Tools** | **Performance Monitor** in more recent versions, with a lot of counters to add. How exactly it operates depends on the Windows version; in older versions, we can click on the **+** icon in the child toolbar, or press *Ctrl* + *I* to see available counters:



Add Counters

Use local computer counters [✓ Select counters from computer:]

\\windows

Performance object: Processor

All counters [✓ Select counters from list:]

% C2 Time
% C3 Time
% DPC Time
% Idle Time
% Interrupt Time
% Privileged Time
[✓ % Processor Time]

All instances [✓ Select instances from list:]

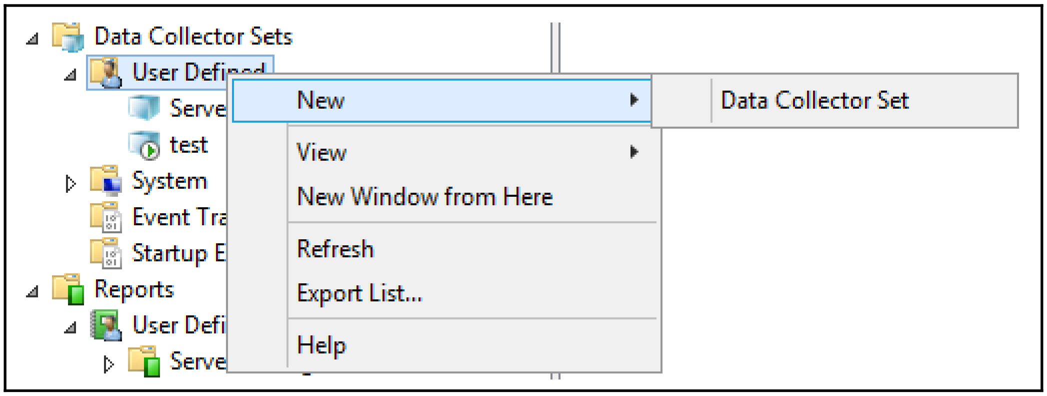
[✓ Total]
0
1

In this dialog, we can gather the information required to construct a performance-counter string. First, the string has to start with a backslash, `\`. The **Performance object** drop-down follows; in this case, **Processor**. Then we have to include the desired instance in parentheses, which makes our string so far `\Processor(_Total)` (notice the leading underscore before `Total`). The `counter` string is finished by adding an individual `counter` string from the **Select counters from list** radio button, again separated by a backslash. So the final performance-counter string looks like this:

```
\Processor(_Total)\% Idle Time
```

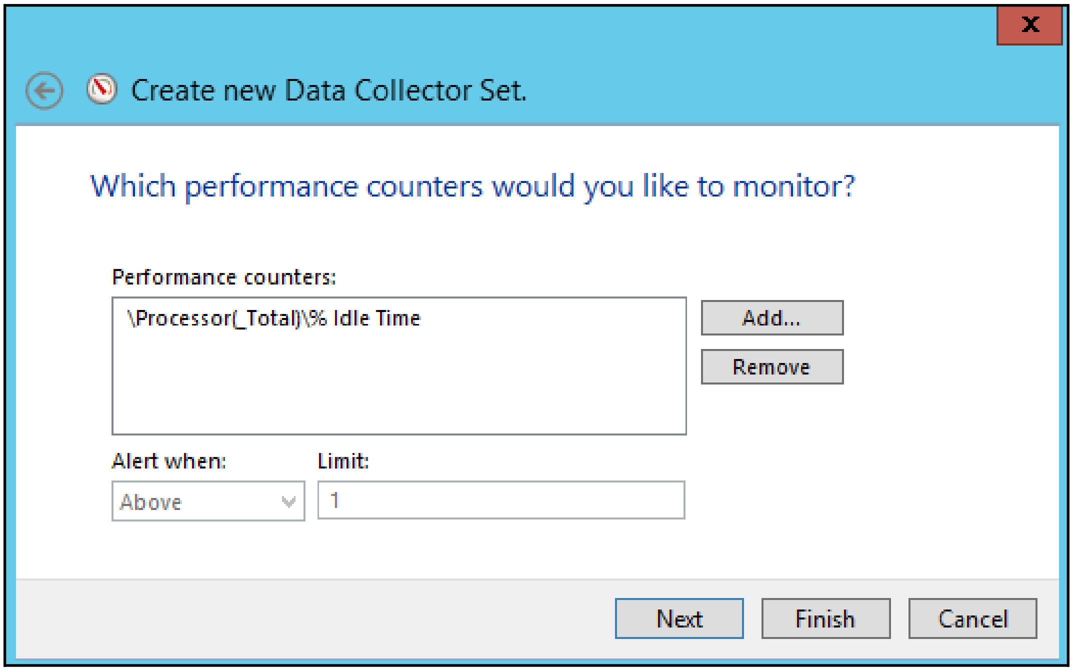
In recent Windows versions, we expand **Data Collector Set**:

1. Right-click on **User Defined**, and choose **New** | **Data Collector Set**:



2. In the resulting window, enter a name for the data collector set, choose **Create manually**, and click on **Next**
3. Choose **Performance Counter Alert**, and click on **Next** again, then click on **Add**
4. Expand **Processor** and click on **% Idle Time**, then click on **Add...**

5. Click on **OK** to see the constructed performance-counter string:



Create new Data Collector Set.

Which performance counters would you like to monitor?

Performance counters:

\Processor(_Total)\% Idle Time

Alert when: Above

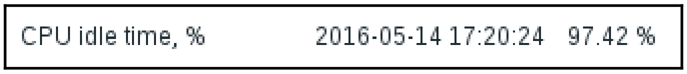
Limit: 1

Now that we have constructed it, *what do we do with it?* Create an item, of course.

Back in the frontend, navigate to **Configuration** | **Hosts**, click on **Items** next to the **Windows host**, and click on **Create item**. Fill in these values:

- **Name**: `CPU idle time, %`
- **Key**: This is where things get more interesting, although the principle is quite simple—the `perf_counter` key has to be used with the performance-counter string, like the one we constructed before as a parameter; thus, enter `perf_counter[\Processor(_Total)\% Idle Time]` here
- **Type of information**: **Numeric (float)**
- **Units**: `%`

When you are done, click on the **Add** button at the bottom. This item should show us the total time all CPUs spend idling on the machine, so let's look at **Monitoring** | **Latest data**. We can see that the data is directly fetched from the built-in performance counter:



```
CPU idle time, %          2016-05-14 17:20:24   97.42 %
```

Looking at the list of available performance objects and corresponding counters in Windows, we can see many different metrics. Navigating this window is cumbersome at best, thanks to small widgets, no proper filtering or searching capabilities, and the fact that constructing the required string to be used as a key is a manual typing job, as entries can't be copied. Luckily, there's a solution available—the `typeperf.exe` command-line utility. To see how it can help us, execute the following:

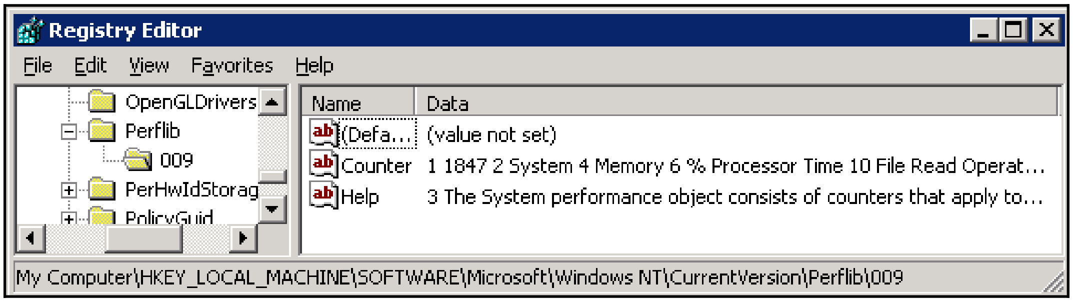
```
C:\zabbix>typeperf -qx > performance_counters.txt
```

This will direct all output of this command to be saved in the `performance_counters.txt` file. Open that file with a text editor and observe the contents. You'll see lots and lots of performance-counter strings, covering various software and hardware information. There is no need to struggle with that clumsy dialog anymore; we can easily search for and copy these strings.

Using numeric references for performance counters

If you have a localized Windows installation, you have probably noticed by now that all performance counters are in the localized language, not in English. This becomes especially cumbersome to handle if you have to monitor several Windows machines with different locales configured for them. For example, a counter that, on an English Windows installation, is `\System\Processes` ,would be `\Système\Processes` in a French one. *Would it be possible to use some other, more universal, method to refer to the performance counters?* Indeed, it would; we can use numeric references, but first, we have to find out what they are.

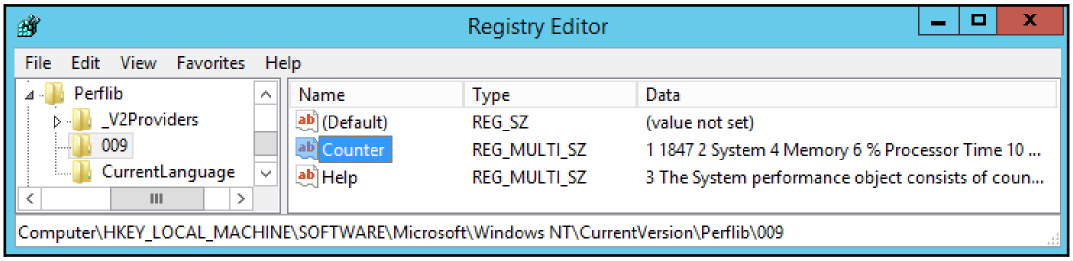
Launch `regedit` and look for the `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib` key. Under this key, you'll see one or more entries, with one being `009`, which is the entry for the English language. Select this entry and pay attention to the **Counter** key, which has suspiciously similar contents to performance-counter names. Expect to see something like this in older versions of Windows:



Name	Data
(Defa...	(value not set)
Counter	1 1847 2 System 4 Memory 6 % Processor Time 10 File Read Operat...
Help	3 The System performance object consists of counters that apply to...

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib\009

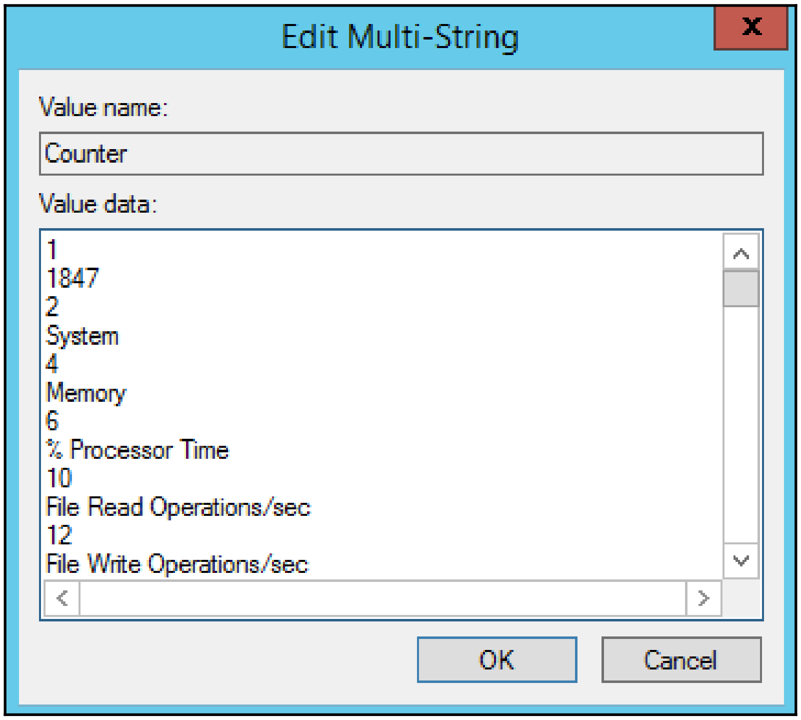
You would see something like this in a more recent version:



Name	Type	Data
(Default)	REG_SZ	(value not set)
Counter	REG_MULTI_SZ	1 1847 2 System 4 Memory 6 % Processor Time 10 ...
Help	REG_MULTI_SZ	3 The System performance object consists of coun...

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib\009

Double-click this value to see its contents in a somewhat more manageable form:



```
Counter
```

```
1
1847
2
System
4
Memory
6
% Processor Time
10
File Read Operations/sec
12
File Write Operations/sec
```

Each performance-counter string can be translated into a number. Figuring out exact conversions in this tiny window is hard, so let's copy all the contents and save them to a file, which we'll then be able to search; name it `numeric.txt`. To see how this works, let's translate the performance-counter string we used before: `\Processor(_Total)\% Idle Time`.

First we have to translate the performance object, `Processor`. While it is possible to search these contents in any text editor, it soon becomes cumbersome, especially if we have to translate lots of values. In that case, we can turn to the basic GNU tools, such as `grep`, which you might have installed on the Windows machine—if not, copy this file over to the Zabbix server:

```
$ grep -B 1 "^Processor$" numeric.txt
```

This command will search for a line containing the `Processor` string exactly and will also output the line immediately before it, which contains the numeric ID of this performance object:

```
238
Processor
```

Numeric values might differ between Windows versions, so make sure to use the values found in your file.

If you are using `grep` on the Zabbix server, the saved file might contain Windows-style newlines and you might get no output. In that case, convert the newlines by executing the following:

```
$ sed -i 's/\r//' numeric.txt
```

Now that we have the numeric value for the first part, do the same for the second part of the performance counter:

```
$ grep -B 1 "^% Idle Time$" numeric.txt
1482
% Idle Time
```

We now have numeric values for all parts of the performance counter, except `_Total`. *How can we translate that?* We don't have to—this string is used as-is on all locales. Our resulting performance counter would then look like this:

```
\238(_Total)\1482
```

As we already have an item gathering this information, we won't add another one. Instead, let's test it with the `zabbix_get` utility. On the Zabbix server, execute the following:

```
 $ zabbix_get -s <Windows host IP> -k
"perf_counter[\238(_Total)\1482]"
```

This should return the same data as the `\Processor(_Total)\% Idle Time` key:

```
  99.577165
```

Additional software can add additional performance counters, and numeric values for such counters can differ between systems. In some cases, software modifies existing performance counters, such as adding the firewall software vendor's name to a network interface.

Using aliases for performance counters

Another method to unify item keys that are using Zabbix configurations (so that a single template could be used for all hosts) is to specify performance-counter aliases. To do that, add an `Alias` directive to the Zabbix agent configuration file. For example, if we wanted to refer to the performance counter we used, `\Processor(_Total)\% Idle Time`, as `cpu.idle_time`, we would add the following:

```
Alias = cpu.idle_time:perf_counter[\Processor(_Total)\% Idle Time]
```

Don't forget to restart the agent after making the changes.

On systems with a different locale, the `Alias` entry would use a different performance-counter, but from now on, we can use the same item key for all systems: `cpu.idle_time`.

Averaging performance counters over time

The Zabbix agent has another Windows-specific feature: it can gather performance-counter values and return the average. This way, we can smooth out counters that return data for the last second and reduce the chance of missing abnormal data. For example, we could add a line such as this in the agent demon configuration file:

```
PerfCounter = disk.writes,"\PhysicalDisk(_Total)\Disk Writes/sec",300
```

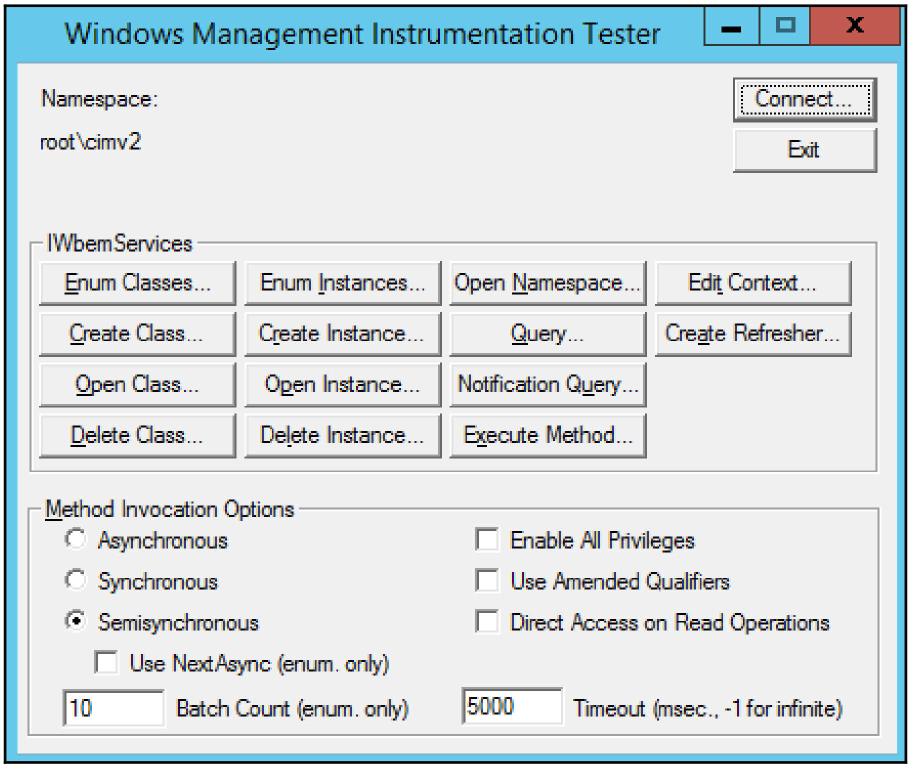
Based on this, the agent will collect the values from that performance counter every second and compute the average over five minutes. We could then query the agent once every five minutes and get an accurate idea of what the average writes per second were. If we didn't use averaging, we would only get the data for the last second once every five minutes.

Querying Windows Management Instrumentation (WMI)

Besides built-in support for performance counters, the Zabbix agent also supports WMI queries.

Zabbix supports WMI through the Zabbix agent—remote WMI is not supported at this time.

To extract some useful information, we need a WMI query, and we might want to test the queries quickly—that can be done in Windows or by using the Zabbix agent. On the Windows side, the `wbemtest.exe` utility can be used. When launching it, click on **Connect...**, accept the default namespace of `root\cimv2`, and click on **Connect...** again. Then, in a dialog such as this, click on **Query...**, as shown in the following screenshot:



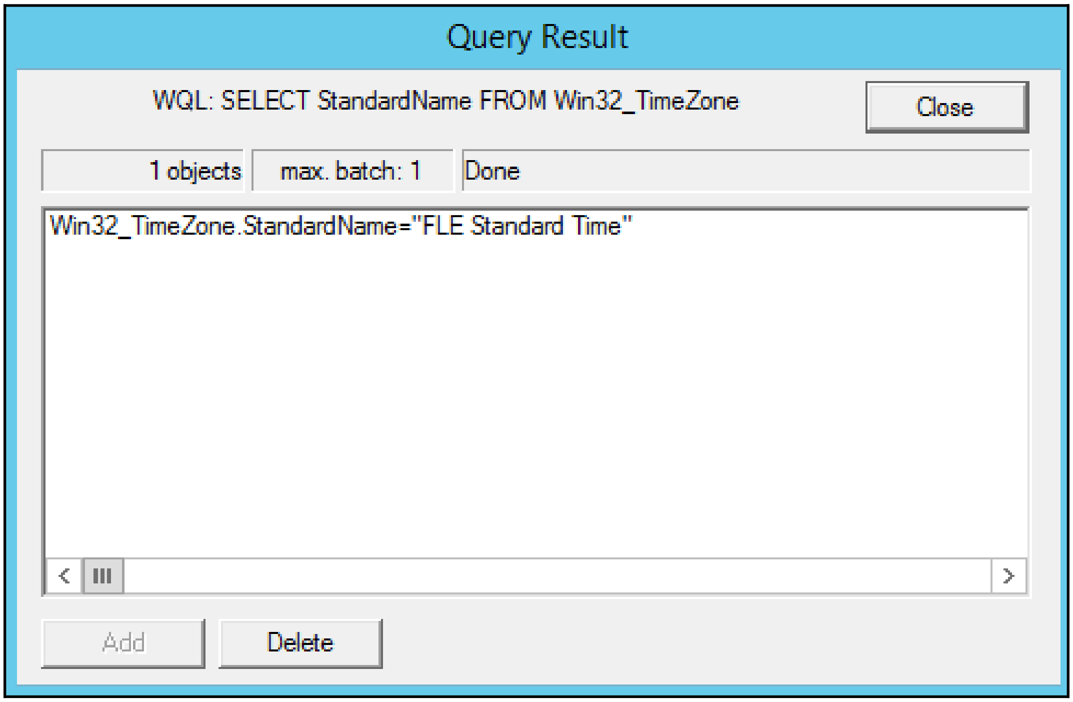
Namespace: root\cimv2

( ) Asynchronous
( ) Synchronous
(●) Semisynchronous
[ ] Use NextAsync (enum. only)
10 Batch Count (enum. only)
[ ] Enable All Privileges
[ ] Use Amended Qualifiers
[ ] Direct Access on Read Operations
5000 Timeout (msec., -1 for infinite)

You can enter complete queries here. For example, we could ask for the current time zone with a query:

```
SELECT StandardName FROM Win32_TimeZone
```

The following screenshot shows the output of the preceding command:



WQL: SELECT StandardName FROM Win32_TimeZone

1 objects | max. batch: 1 | Done

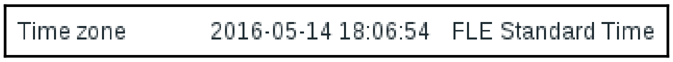
```
Win32_TimeZone.StandardName="FLE Standard Time"
```

An alternative way to test such queries through the Zabbix agent would be with the `zabbix_get` utility, discussed in Chapter 3, *Monitoring with Zabbix Agents and Basic Protocols*.

With the query available, we can proceed with creating an item. Navigate to **Configuration** | **Hosts**, click on **Items** next to **Windows host**, and click on **Create item**. Fill in the following:

- **Name**: `Time zone`
- **Key**: `wmi.get[root\cimv2,SELECT StandardName FROM Win32_TimeZone]`
- **Type of information**: **Character**
- **Update interval**: `300`

The key here was `wmi.get`, the first parameter was namespace, and the second parameter was the query itself. We don't expect the timezone to change that often, so we increased the update interval a bit—normally, we would use an even larger interval, but we will want the first value to come in soon enough. When done, click on the **Add** button at the bottom. Check **Monitoring** | **Latest data**—in five minutes, the value should be there:

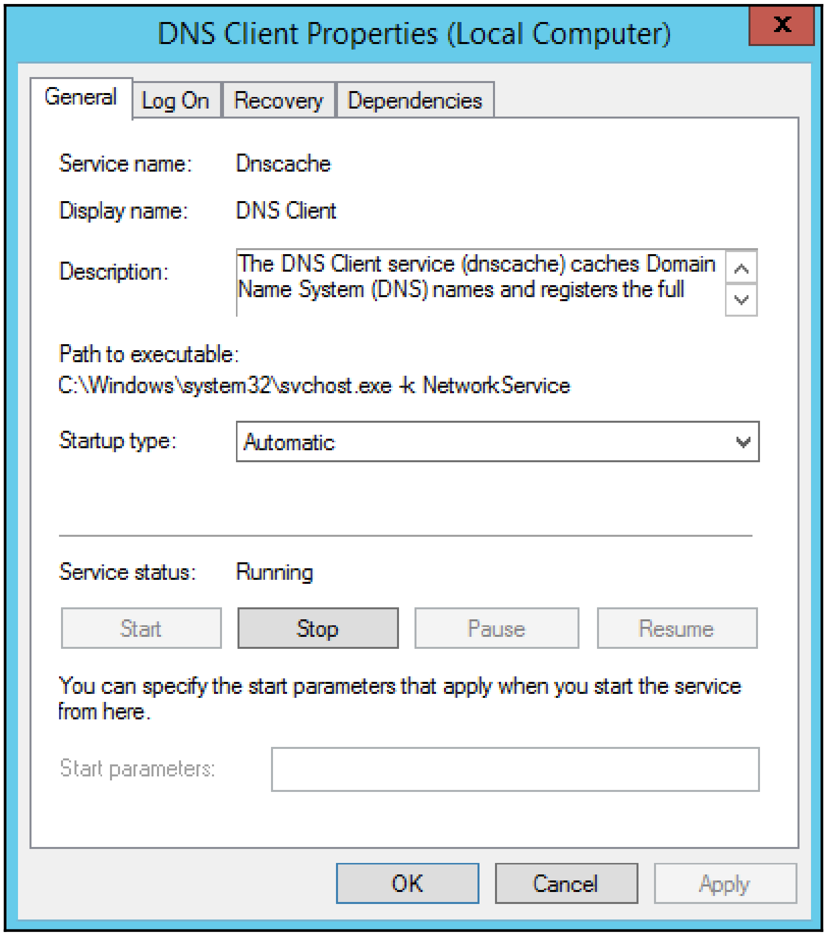


```
Time zone          2016-05-14 18:06:54    FLE Standard Time
```

This way, we can monitor any output from the WMI queries—but note that a single value should be returned; multiple values are not supported. If multiple values are returned, only the first value will be processed.

Monitoring Windows services

There's yet another item category that is Windows-specific: a dedicated key for Windows service state monitoring. Let's try to monitor a service now. First, we have to figure out how to refer to this service. For that, open the services list and then open up the details of a service—let's choose **DNS Client**:



Service name: Dnscache
Display name: DNS Client
Description: The DNS Client service (dnscache) caches Domain Name System (DNS) names and registers the full
Path to executable:
C:\Windows\system32\svchost.exe -k NetworkService
Startup type: Automatic
Service status: Running
You can specify the start parameters that apply when you start the service from here.
Start parameters:

Look at the top of this tab. **Service name** is the name we will have to use, and we can see that it differs noticeably from the display name; instead of using **DNS Client**, the name is **Dnscache**.

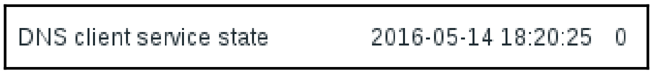
Let's create the item now. Navigate to **Configuration** | **Hosts**, click on **Items** next to the **Windows host**, then click on **Create item**. Enter these values:

- **Name:** `DNS client service state`
- **Key:** `service.info[Dnscache]`

Service names are case-insensitive.

The key used here, `service.info`, is new in Zabbix 3.0. Older versions of Zabbix used the `service_state` key. This key is deprecated but still supported, and you are likely to see it in older Zabbix installations and templates. The `service.info` key has more parameters—for the complete documentation, consult the Zabbix manual.

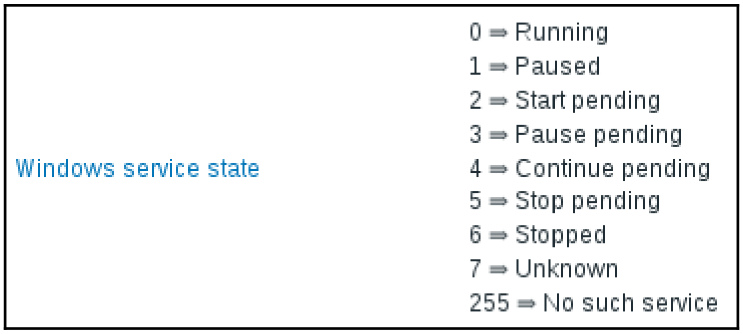
When done, click on the **Add** button at the bottom, open **Monitoring** | **Latest data**, and look for our newly-added item:



DNS client service state	2016-05-14 18:20:25	0

So data is gathered, and the state is `0`. That's probably normal, but *how can we know what the state number means?*

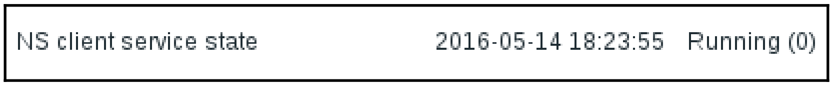
Back in **Configuration** | **Hosts**, click on **Items** next to **Windows host** and click on **DNS client service state** in the **Name** column. Look at our old friend, the **Show value** property. Click on the **Show value mappings** link and examine the mapping near the bottom of the list:



Windows service state	0 ⇒ Running 1 ⇒ Paused 2 ⇒ Start pending 3 ⇒ Pause pending 4 ⇒ Continue pending 5 ⇒ Stop pending 6 ⇒ Stopped 7 ⇒ Unknown 255 ⇒ No such service

It turns out there's already a predefined mapping for Windows service states available.

Close this window and choose **Windows service state** in the **Show value** drop-down, then click on **Update**. Back in **Monitoring** | **Latest data**, verify that the service state is now displayed in a much more user-friendly way:



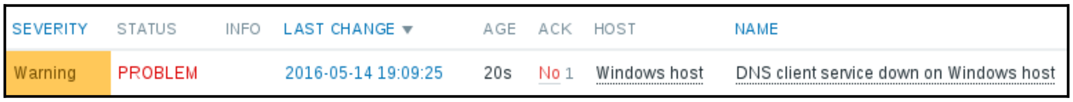
NS client service state 2016-05-14 18:23:55 Running (0)

Now we will be able to easily identify different service states in the frontend. With the item in place, let's also create a trigger that will alert us when this service has stopped.

Go to **Configuration** | **Hosts**, click on **Triggers** next to **Windows host**, and click on **Create trigger**. Enter `DNS client service down on {HOST.NAME}` in the **Name** field, then click on **Add** next to the **Expression** field. Click on **Select** next to the **Item** field, choose **DNS client service state**, and click on **Insert**. But wait, the value of `0` was for when the service was running; we should actually test for the value not being `0`. We just avoided using the drop-down function that changes the insert expression, as follows:

```
{Windows host:service.info[Dnscache].last()}<>0
```

Change the severity to **Warning** and click on the **Add** button at the bottom. Unless this is a production system, it should be pretty safe to stop this service—do so, and observe **Monitoring** | **Triggers**; select **Windows servers** in the **Group** drop-down. Zabbix should now warn you that this service is down:



SEVERITY	STATUS	INFO	LAST CHANGE ▼	AGE	ACK	HOST	NAME
Warning	PROBLEM		2016-05-14 19:09:25	20s	No 1	Windows host	DNS client service down on Windows host

Checking automatic services

Sometimes we are not interested in the exact details of every service, and we might have to configure an item and trigger for each of them manually. Instead, we might want to see a high-level overview; for example, whether any of the services that are started automatically have stopped. Zabbix provides an item that allows you to make such a comparison very easily: services. It allows us to retrieve lists of services based on different parameters, including ones that should be started automatically and are stopped. *How can we use this?*

An item should be added with the following key:

```
services[automatic,stopped]
```

For a list of all supported services, key parameters, consult the Zabbix manual.

This will take care of getting the required data. Whenever a service that is set to start automatically is stopped, it will be listed in the data from this item.

It is also possible that on some Windows versions there will be services that are supposed to start up automatically and shut down later. In this case, they would appear in the listing and break our monitoring. Luckily, Zabbix has a solution for such a problem, too—we can add a third parameter to this key and list services to be excluded from this check. For example, to exclude the `RemoteRegistry` and `sppsvc` services, the key would be the following:

```
services[automatic,stopped,"RemoteRegistry,sppsvc"]
```

Notice how the services to be excluded are comma-delimited, and the whole list is included in double quotes.

If the list of such services is different between hosts, consider using a user macro to hold the service list. We discussed user macros in Chapter 8, *Simplifying Complex Configurations with Templates*.

But *how do we check that everything is good in a trigger?* If the list is empty, the Zabbix agent returns `0`. As a result, by simply checking whether the last value was zero, we can trigger when an automatically-started service is stopped. A trigger expression for such a check would be the following:

```
{Windows host:services[automatic,stopped].last()}<>0
```

Of course, you can apply a method—such as using the `count()` function—to only fire the trigger after it has been non-zero for more than a single check:

```
{Windows host:services[automatic,stopped].count(#3,0)}=0
```

Such a trigger expression will only fire if there has been at least one such stopped service in all of the last three checks.

Service discovery

The preceding method just tells you that some service that was supposed to be running has stopped. To see which service that is, we'd have to look at the item values. We can actually monitor all services individually, as Zabbix has supported **Windows service discovery** since version 3.0. Let's discover all Windows services and monitor some parameter on all of them—we can choose the service description here.

Navigate to **Configuration | Hosts**, click on **Discovery** next to **Windows host**, and click on **Create discovery rule**. Fill in the following:

- **Name**: `Windows service discovery`
- **Key**: `service.discovery`
- **Update interval**: `300`

We used a built-in agent key and increased the update interval. In production, it is probably a good idea to increase the interval even more; an average default interval for discovery rules of one hour is likely a good idea. When done, click on the **Add** button at the bottom. We have the rule itself; now we need some prototypes—click on **Item prototypes**, then click on **Create item prototype**. Before we fill in the data, it would be useful to know what this discovery item returns—an example for one service is as follows:

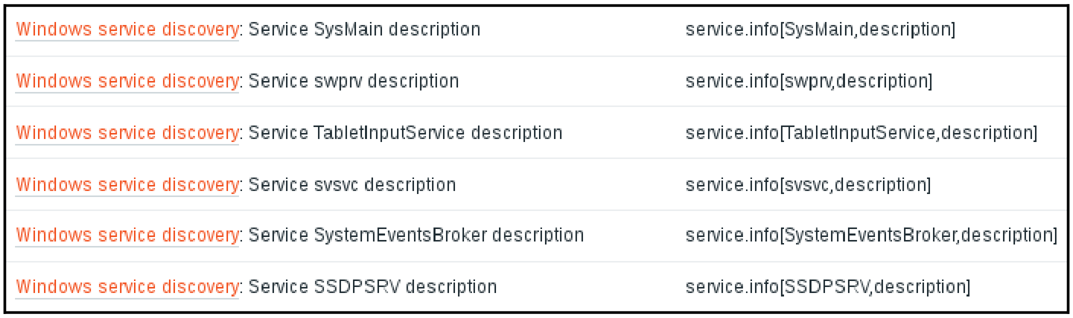
```
{
    "{#SERVICE.STARTUP}" : 0,
    "{#SERVICE.DISPLAYNAME}" : "Zabbix Agent",
    "{#SERVICE.DESCRIPTION}" : "Provides system monitoring",
    "{#SERVICE.STATENAME}" : "running",
    "{#SERVICE.STARTUPNAME}" : "automatic",
    "{#SERVICE.USER}" : "LocalSystem",
    "{#SERVICE.PATH}" : "\"C:\\zabbix\\zabbix_agentd.exe\" --config
\"c:\\zabbix\\zabbix_agentd.win.conf\"",
    "{#SERVICE.STATE}" : 0,
    "{#SERVICE.NAME}" : "Zabbix Agent"
}
```

The Zabbix agent can be queried for the raw LLD data using `zabbix_get`. We discussed **low-level discovery** (**LLD**) in more detail in Chapter 11, *Automating Configuration*.

This snippet also shows what other things we could monitor for each service. For now, we want to extract descriptions for all services, but to add the items we need the actual service names. Although the description is available here, we will query it in the item, so for item prototypes it will actually be the `{#SERVICE.NAME}` macro. With this knowledge, we are ready to fill in the item prototype form:

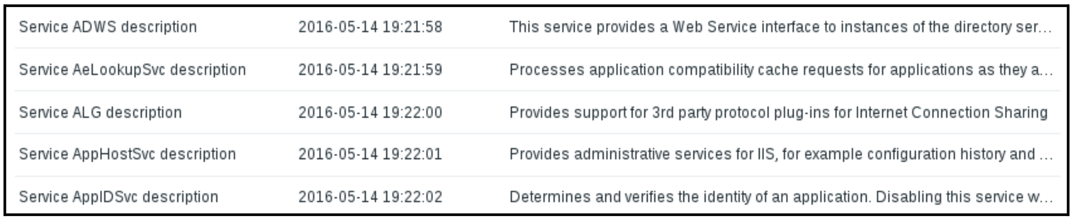
- **Name:** `Service $1 description`
- **Key**: `service.info[{#SERVICE.NAME},description]`
- **Type of information**: **Character**
- **Update interval**: `300`

When done, click on the **Add** button at the bottom. With our discovery running every five minutes, it might take up to five minutes for this prototype to generate actual items, and then it would take up to six minutes for these items to get their first value—the added time of configuration cache update and item interval. First, go to item configuration for the Windows host. After a while, our discovery rule should add the items:



Windows service discovery: Service SysMain description	service.info[SysMain,description]
Windows service discovery: Service swprv description	service.info[swprv,description]
Windows service discovery: Service TabletInputService description	service.info[TabletInputService,description]
Windows service discovery: Service svsvc description	service.info[svsvc,description]
Windows service discovery: Service SystemEventsBroker description	service.info[SystemEventsBroker,description]
Windows service discovery: Service SSDPSRV description	service.info[SSDPSRV,description]

There will likely be a fairly large number of such items. By visiting **Monitoring** | **Latest data**, after a few more minutes, we should see descriptions for all services:



Service ADWS description	2016-05-14 19:21:58	This service provides a Web Service interface to instances of the directory ser...
Service AeLookupSvc description	2016-05-14 19:21:59	Processes application compatibility cache requests for applications as they a...
Service ALG description	2016-05-14 19:22:00	Provides support for 3rd party protocol plug-ins for Internet Connection Sharing
Service AppHostSvc description	2016-05-14 19:22:01	Provides administrative services for IIS, for example configuration history and ...
Service AppIDSvc description	2016-05-14 19:22:02	Determines and verifies the identity of an application. Disabling this service w...

A more common approach would be to monitor the current service state or its startup configuration—anything the `service.info` key supports should be possible.

We can also use any of the LLD macros to filter the discovered services. For example, via `filtering` for `{#SERVICE.STARTUP}`, we could discover only the services that are configured to start up automatically (value 0), or start automatically with a delay (value 1).

Windows event-log monitoring

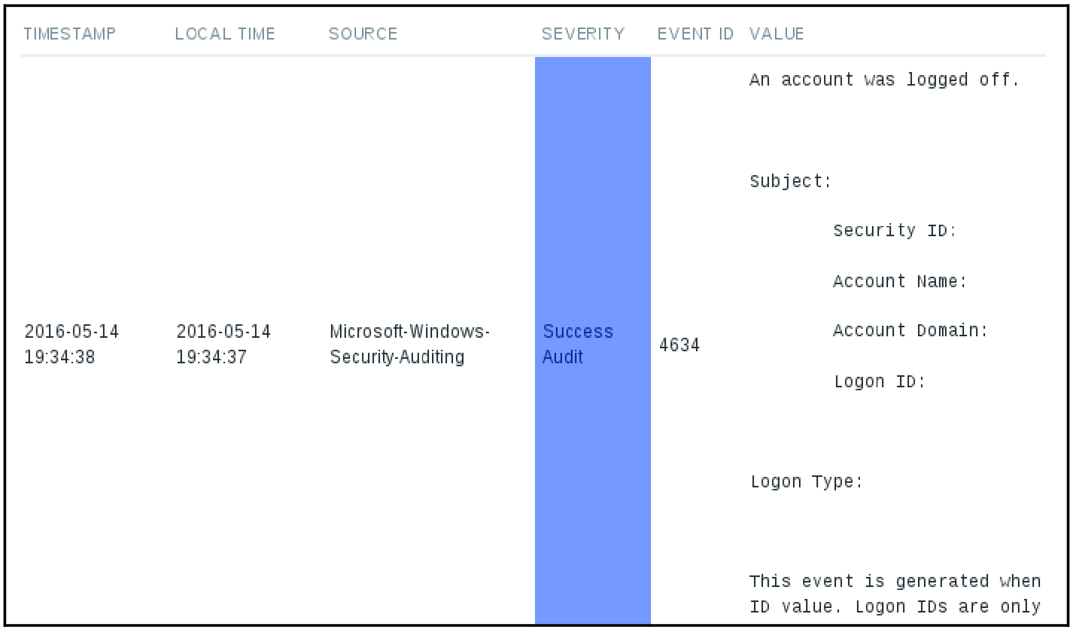
Zabbix supports log-file monitoring on Windows as well—the topics we discussed in Chapter 10, *Advanced Item Monitoring,* still apply. But on Windows, there is also a specialized logging subsystem, and Zabbix does offer built-in event log system support. Windows has various event-log categories, and we could monitor the security event log. Other common logs are system and application, and there will be more logs in recent versions of Windows.

For now, let's head to **Configuration** | **Hosts**, click on **Items** next to **Windows host**, and click on **Create item**. Fill in the following:

- **Name**: `Windows $1 log`
- **Type**: **Zabbix agent (active)**
- **Key**: `eventlog[Security,,,,,,skip]`
- **Type of information**: **Log**
- **Update interval**: `1`

Event-log monitoring on Windows works as an active item, just like normal log-file monitoring.

That's six commas in the item key. When done, click on the **Add** button at the bottom. The last parameter we specified here, `skip`, will prevent the agent from reading all of the security log—a pretty good idea for systems that have been around for some time. Visit **Monitoring** | **Latest data** and click on **History** for the **Windows Security log** item:



TIMESTAMP	LOCAL TIME	SOURCE	SEVERITY	EVENT ID	VALUE
2016-05-14 19:34:38	2016-05-14 19:34:37	Microsoft-Windows-Security-Auditing	Success Audit	4634	An account was logged off. Subject: Security ID: Account Name: Account Domain: Logon ID: Logon Type: This event is generated when ID value. Logon IDs are only

If no values appear, sign into the Windows system—that should generate some entries in this log.

A few notable differences, compared to normal log-file monitoring, include automatic data-population in the **LOCAL TIME** column, as well as source, severity, and the event ID being stored. Actually, we can filter by some of these already at the agent level; we don't have to send all entries to the server. Let's discuss some of the item's key parameters in a bit more detail. The general key syntax is this:

```
eventlog[name,<regexp>,<severity>,<source>,<eventid>,<maxlines>,<mode>
]
```

The second parameter, `regexp`, operates the same as in normal log-file monitoring—it matches a regular expression against the log entry. The `maxlines` and `mode` parameters work the same as they do for the `log` and `logrt` item keys. The `severity`, `source`, and `eventid` parameters are specific to the `eventlog` key, and they are all regular expressions to be matched against the corresponding field. This way, we can filter `eventlog` quite extensively on the agent side, but people make a somewhat common mistake sometimes—they forget that these are regular expressions, not exact match strings. *What does that mean?* Well, the following item key would not only match events with the ID of `13`, as follows:

```
eventlog[Security,,,,13]
```

It would also match events with IDs of `133`, `1333`, and `913`. To match `13`, and `13` only, we'd have to anchor the regular expression:

```
eventlog[Security,,,,^13$]
```

Remember that it is true for the `severity` and `source` parameters as well—while they are less likely to match an unintended value, you should always make sure the expression is anchored if an exact match is desired.

Summary

In this chapter, we explored various things that were either different on Windows, or things that Zabbix explicitly supports on Windows.

We installed the Zabbix agent as a Windows service and verified that, in many ways, it works exactly the same as the Linux agent. Then we moved to Windows-specific feature support for the following:

- Performance counters
- WMI using the Zabbix agent
- Windows services, including the ability to automatically discover them
- The event log system

Not only did we discuss details and potential issues for all of these, we also monitored some data using each of these features. Coupled with the generic monitoring and reporting knowledge that we have now, this should allow us to efficiently monitor Windows installations as well.

Having explored quite a lot of lower-level configurations, in the next chapter, we will look at a more business-oriented aspect—SLA monitoring. Zabbix allows us to create an IT service tree, assign triggers that depict service availability, and calculate how much of an adherence to the SLA that is.