

NetXMS User Manual

Table of Contents

1	Introduction.....	4
1.1	About NetXMS.....	4
1.2	What you can do with NetXMS.....	4
1.3	About this manual.....	4
2	Basic Concepts.....	5
2.1	Overview of System Architecture.....	5
2.2	Objects.....	6
2.3	DCI (Data Collection Items).....	7
2.4	Thresholds.....	8
2.5	Events and Alarms.....	8
3	Initial Configuration and Network Discovery	9
3.1	Configure Your NetXMS Server.....	9
3.2	Secure your installation.....	9
3.3	Discover the network.....	10
3.4	Manually add nodes.....	10
4	Objects.....	11
4.1	Overview.....	11
4.2	Top Level Objects.....	11
4.3	Subnet objects.....	12
5	Data Collection.....	13
5.1	How data collection works.....	13
5.2	DCI configuration.....	13
5.2.1	Basic configuration.....	13
5.2.1.1	Description.....	13
5.2.1.2	Parameter.....	13
5.2.1.3	Origin.....	13
5.2.1.4	Data Type.....	14
5.2.1.5	Polling Interval.....	14
5.2.1.6	Use Advanced Schedule.....	14
5.2.1.7	Associate with cluster resource.....	14
5.2.1.8	Retention Time.....	15
5.2.1.9	Status.....	15
5.2.2	Data Transformations.....	15
5.2.3	Thresholds.....	15
5.2.3.1	Overview.....	15
5.2.3.2	Instance.....	16
5.2.3.3	Threshold Processing.....	16
5.2.3.4	Threshold Configuration.....	17
5.2.3.5	Thresholds and Events.....	18
5.2.4	Push parameters.....	19
5.3	Templates.....	19
5.3.1	What is templates.....	19
5.3.2	Creating template.....	20
5.3.3	Configuring templates.....	20
5.3.4	Applying template to node.....	20
5.3.5	Removing template from node.....	20
5.3.6	Macros in template items.....	20
5.4	Working with collected data.....	21
5.4.1	View collected data in graphical form.....	21
5.4.2	View collected data in textual form.....	22
5.4.3	Export DCI data.....	23
6	Event Processing.....	24
6.1	Event Processing Overview.....	24
6.2	Event Processing Policy.....	24
6.3	Alarms.....	25

6.3.1 Alarms Overview.....	25
6.3.2 Generating Alarms.....	26
6.3.3 Automatic Alarm Termination.....	26
6.4 Situations.....	27
6.4.1 Situations Overview.....	27
7 Troubleshooting.....	28
7.1 Server problems.....	28
7.2 Agent problems.....	28
8 Complete Reference.....	29
8.1 Server Configuration.....	29
8.1.1 File: netxmtd.conf.....	29
8.1.2 Table in Database: config.....	29
8.2 Agent Configuration.....	35
8.2.1 File: nxagentd.conf.....	35
8.3 Web Server (nxhttpd) Configuration.....	38
8.3.1 File nxhttpd.conf.....	38
8.4 Command Line Tools.....	39
8.4.1 Local Server Administration Tool (nxadm).....	39
8.4.2 Agent GET Tool (nxget).....	40
8.4.3 NetXMS Database Manager (nxdbmgr).....	42
8.5 NetXMS Scripting Language (NXSL).....	43
8.5.1 Formal Grammar.....	43
8.5.2 Built-in Functions.....	45
8.5.2.1 abs.....	45
8.5.2.2 AddrInRange.....	45
8.5.2.3 AddrInSubnet.....	45
8.5.2.4 classof.....	46
8.5.2.5 gmtime.....	46
8.5.2.6 left.....	46
8.5.2.7 length.....	47
8.5.2.8 localtime.....	47
8.5.2.9 lower.....	47
8.5.2.10 max.....	48
8.5.2.11 min.....	48
8.5.2.12 pow.....	48
8.5.2.13 right.....	48
8.5.2.14 SecondsToUptime.....	48
8.5.2.15 strftime.....	49
8.5.2.16 substr.....	50
8.5.2.17 time.....	50
8.5.2.18 typeof.....	50
8.5.2.19 upper.....	50
8.5.3 Functions Available in DCI Transformation Scripts.....	51
8.5.3.1 FindDCIByDescription.....	51
8.5.3.2 FindDCIByName.....	51
8.5.3.3 GetDCIValue.....	51
8.5.4 Functions Available in Event Processing Scripts.....	51
8.5.4.1 FindSituation.....	51
8.5.4.2 GetSituationAttribute.....	52
8.6 Macros for Event Processing.....	53

1 Introduction

1.1 About NetXMS

NetXMS is new and rapidly developing monitoring system, released under GPL2 license. It can be used for monitoring entire IT infrastructure, starting with SNMP-capable hardware (like switches and routers) and ending with applications on your servers. NetXMS is an extremely reliable and powerful monitoring system, enabling you to improve your network availability and service levels. The system has three-tier architecture: the information is collected by monitoring agents (either our own high-performance agents or SNMP agents) and delivered to monitoring server for processing and storage. Network administrator can access collected data using Windows-based Management Console, WEB Interface or Management Console for PocketPC. Having been designed with flexibility and scalability in mind, NetXMS features a wide range of supported platforms, leaving you the freedom of choice of platform(s) for your network. NetXMS Server, the core system, is currently available for Windows NT/2000/2003/XP, Linux, Solaris, AIX, HP-UX, and FreeBSD. High-performance modular monitoring Agents are available for the same platforms as well as for OpenBSD, NetBSD, Novell NetWare, and Nokia IPSO. NetXMS currently supports the following databases: MySQL, PostgreSQL, SQLite, Microsoft SQL and Oracle.

1.2 What you can do with NetXMS

NetXMS can help you accomplish many tasks in network management.

With NetXMS you can:

- monitor status of your network devices;
- monitor status of your hosts;
- monitor status of applications running on your servers;
- collect performance data from your network devices;
- collect performance data from servers;
- store collected data for later analysis;
- discovery network IP topology;
- automatically discover new hosts and devices;
- notify system administrator(s) about problems via e-mail or SMS.

1.3 About this manual

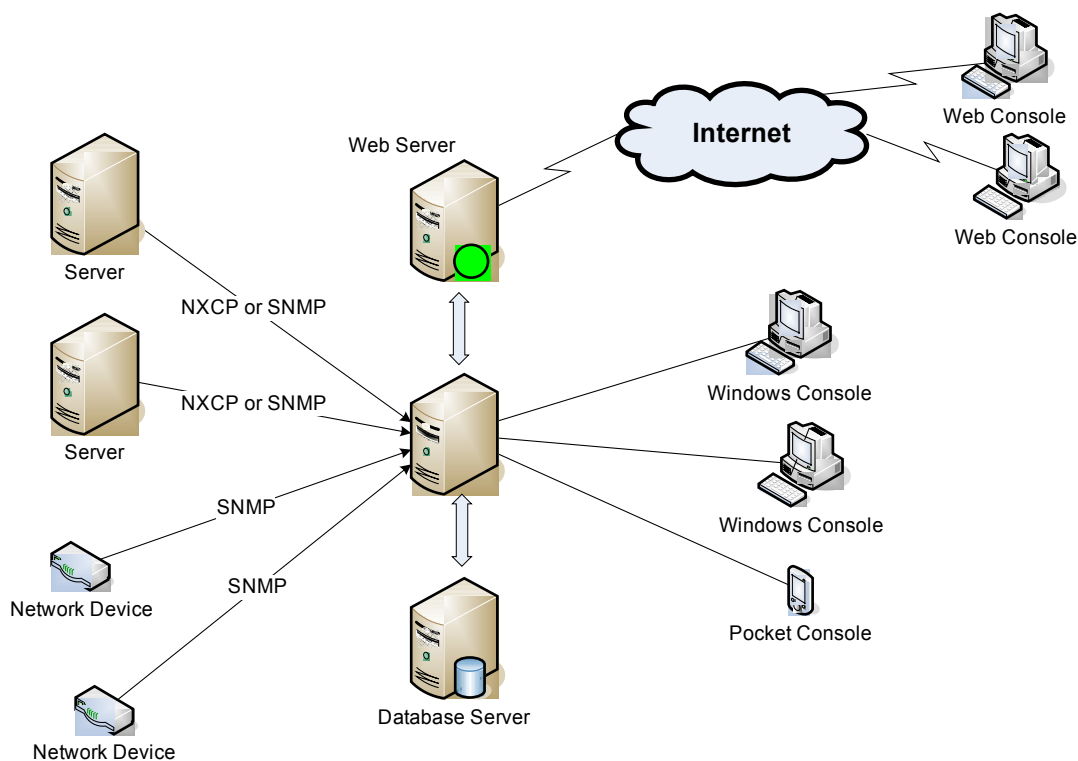
This manual is intended for both NetXMS administrators and users, and provides all information necessary to successfully operate NetXMS. It's assumed that you are using NetXMS Console for Windows (nxcon.exe).

2 Basic Concepts

2.1 Overview of System Architecture

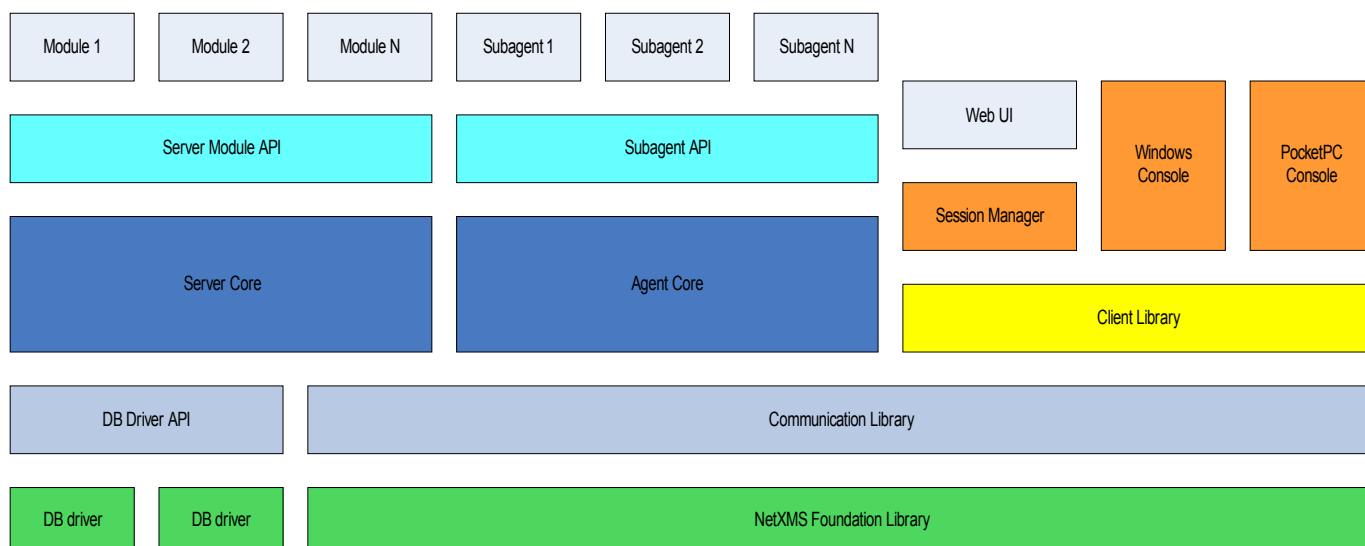
The system has three-tier architecture: the information is collected by monitoring agents (either NetXMS native agents or SNMP agents) and delivered to monitoring server for processing and storage. Network administrator can configure system and access collected data using Windows-based Management Console, WEB Interface or Management Console for PocketPC. You can see an overview of NetXMS architecture on Figure 1.

Figure 1: NetXMS architecture overview



All collected data and system configuration is stored in the SQL database. You can choose Oracle, Microsoft SQL Server, PostgreSQL, MySQL, or SQLite as your database engine. Database server can be installed on the same physical machine, or be a separate server (as shown on Figure 1). NetXMS has its own, lightweight web server, which does not depend on any general-purpose web server engines. It is a separate component and can be installed on the same physical machine as NetXMS server, or on a remote server.

The system was designed to be easily extendable; so all three tiers — server, agent, and client — have modular structure. Figure 2 shows main software layer of NetXMS system.

Figure 2: NetXMS main software layer

All system components use two libraries: NetXMS Foundation Library and Communication Library. These libraries provide communication between all system components. The server also has an underlying DB Driver API layer, which provides uniform database engine interface by using database drivers. This approach allows developers to add support for new database engines in a matter of days without changing or even recompiling server code.

On top of server core, another interface, Server Module API, provides a uniform way to create server extensions. The same approach is used with NetXMS agent, which has a Subagent API on top of agent core.

Portable NetXMS Client Library provides consistent API for accessing and managing NetXMS server. This library has been ported to Windows, UNIX, and Pocket Windows platforms. All client components of NetXMS – management console, alarm viewer, web server, and console for Pocket PC – use this library. If you wish to write your own client application for NetXMS or need to integrate existing system with NetXMS, you can use our client library. Please consult *NetXMS Client Library Programmer's Manual* for detailed information.

2.2 Objects

All network infrastructure monitored by NetXMS inside monitoring system represented as a set of *objects*. Each object represents one physical or logical entity (like host or network interface), or group of them. Objects organized into hierarchical structure. There are 12 different object classes:

Entire Network	Abstract object representing root of IP topology tree. All subnet objects located under it. System can have only one object of this class.
Subnet	Object representing IP subnet. Typically objects of this class created automatically by the system to reflect system's knowledge of IP topology.
Node	Object representing physical host or network device. These objects can be created either manually by administrator or automatically during network discovery process.
Cluster	Object representing cluster consisted of two or more hosts.
Interface	Object representing network interface of node. These objects created automatically by the system during configuration polls.
Network Service	Object representing network service running on a node (like http or ssh).
VPN Connector	Object representing VPN tunnel endpoint. Such objects can be created to add VPN tunnels to network topology known y NetXMS server.

Service Root	Abstract object representing root of your service tree. System can have only one object of this class.
Container	Grouping object which can contain nodes, subnets, clusters, conditions, or other containers. With help of container objects you can build object's tree which represents logical hierarchy of IT services in your organization.
Condition	Object representing complicated condition – like "cpu on node1 is overloaded and node2 is down for more than 10 minutes".
Template Root	Abstract object representing root of your template tree.
Template Group	Grouping object which can contain templates or other template groups.
Template	Data collection template. See <i>Data Collection</i> section for more information about templates.

Every object has set of attributes; some of them are common (like id and name), while other depends on object class – for example, attribute "SNMP community string" have only node objects.

2.3 DCI (Data Collection Items)

Every node can have many parameters, like CPU utilization or amount of free memory, which can be collected by management server, checked for threshold violations, and stored in the database. In NetXMS, parameters configured for collection are called *Data Collection Items* or DCI for short. One DCI represents one node's parameter, and unlimited number of DCIs can be configured for any node.

Each data collection item has various attributes controlling its handling:

- *Description* — a free-form text string describing DCI. It is not used by the server and is intended for better information understanding by operators.
- *Origin* — origin of data (or method of obtaining data). Possible origins are NetXMS agent, SNMP agent, CheckPoint SNMP agent, or Internal (data generated inside NetXMS server process).
- *Name* — name of the parameter of interest, used for making a request to target node. For NetXMS agent it will be parameter name, and for SNMP agent it will be an SNMP OID.
- *Data Type* — data type for a parameter. Can be one of the following: Integer, Unsigned Integer, 64-bit Integer, 64-bit Unsigned Integer, Float (floating point number), or String. Selected data type affects processing of collected data — for example, you cannot use operations like "less than" or "greater than" on strings.
- *Retention Time* — this attribute specifies how long collected data should be kept in database, in days. Minimum retention time is 1 day, and maximum is not limited. However, keeping too many collected values for too long will lead to significant increase of your database size and possible performance degradation.
- *Schedule Type* — type of the collection schedule used; can be either simple or advanced. In a simple mode, values are taken from target at fixed intervals. In an advanced mode, cron-like scheduling table can be used to specify the exact time for polling. This can be useful if, for example, you wish to check the file size every Monday and Friday at 7:00.
- *Polling Interval* — interval in seconds between two polls. Applicable only if simple schedule type is selected.
- *Scheduling Table* — cron-like scheduling table for data collection polls. Applicable only if advanced schedule type is selected.
- *Threshold List* — list of defined thresholds.
- *Instance* — a free-form text string, passed as 6th parameter to events associated with thresholds. You can use this parameter to distinguish similar events related to different instances of the same entity – for example, if you have an event generated when file system is low on free space, you can set *instance* attribute to file system mount point.

2.4 Thresholds

Each threshold is a combination of condition and events pair — if condition becomes true, associated "activation" event generated, and when it's becomes false again, "deactivation" event generated. Thresholds let you take a proactive approach to network management. You can define thresholds for any data collection items that you are monitoring. When setting thresholds, first determine what would constitute reasonable thresholds. To decide on a threshold value, you need to know what is normal value and what is out of range. Only you can decide what is normal behavior for a device on your network. Generally, we recommend that you collect information about a device throughout one complete business cycle, before determining the normal high/low range. Consider collecting values such as error rates, retry limits, collisions, throughput, relation rates, and many more. You also have the possibility to define more than one threshold for a single DCI, which allows you to distinguish between different severity conditions.

2.5 Events and Alarms

Many services within NetXMS gather information and generate *events* that are forwarded to NetXMS Event Queue. Events can also be emitted from agents on managed nodes, or from management applications residing on the management station or on specific network nodes. All events are processed by NetXMS Event Processor one-by-one, according to the processing rules defined in Event Processing Policy. As a result of event processing, some actions can be taken, and event can be shown up as *alarm*. NetXMS provides one centralized location, the Alarm Browser, where the alarms are visible to your team. You can control which events should be considered important enough to show up as alarms. You and your team can easily monitor the posted alarms and take appropriate actions to preserve the health of your network.

Examples of alarms include:

- A critical router exceeded its threshold of traffic volume that you configured in Data Collection.
- The shell script that you wrote gathered the specific information you needed and posted it to the NetXMS as an event.
- One of your mission-critical servers is using its UPS battery power.
- An SNMP agent on a managed critical server forwarded a trap to NetXMS because it was overheating and about to fail.

3 Initial Configuration and Network Discovery

3.1 Configure Your NetXMS Server

You have to set some important server parameters after installation. These parameters are accessible through the **Server Configuration** window in the console. To access the **Server Configuration** window, press F9 or on the **View** menu click **Control Panel** to access the **Control Panel** window and then click the **Server Configuration** icon. To edit a setting, double click on the row in the table or right-click and select **Edit**. The following parameters may need to be changed:

Table 1: NetXMS Server configuration parameters

Parameter	Value to be set
NumberOfStatusPollers	If you plan to monitor large number of hosts, increase this parameter from the default value to approximately 1/10 of host count.
NumberOfConfigurationPollers	If you plan to monitor large number of hosts, increase this parameter from the default value to approximately 1/20 of host count.
NumberOfDataCollectors	If you plan to monitor large number of hosts, increase this parameter from the default value to approximately 1/10 – 1/5 of host number. Use larger value if you plan to gather many DCIs from each host.
RunNetworkDiscovery	If you plan to use automatic network discovery, set this parameter to 1. You may also use Network Discovery item in Control Panel for simplified discovery network configuration.
DiscoveryFilter	If you want NetXMS to discover only specific hosts, set this parameter to the name of filtering script. After installation, NetXMS server has three preconfigured filtering scripts: <ul style="list-style-type: none">• Filter::Agent — will discover only hosts with active NetXMS agent.• Filter::SNMP — will discover only hosts with active SNMP agent.• Filter::AgentOrSNMP — will discover only hosts with either active NetXMS agent or active SNMP agent. You can also create your own filtering scripts. See "Scripting" chapter for more information.
DefaultCommunityString	Set this parameter to default SNMP community string used on your devices. This community string will be used during network discovery process and manual host addition.
EnableSyslogDaemon	Set this parameter to 1 if you want to enable NetXMS built-in syslog server.

After changing these parameters, you have to restart your NetXMS server so changes will take effect. For simplified network discovery configuration you may also use **Network Discovery** option in **Control Panel**.

3.2 Secure your installation

We recommend that you change password for *admin* user immediately after installation to prevent possible unauthorized access to the system. You can change user passwords in the **User Manager**. To access the **User Manager** window, press F9 or on the **View** menu click **Control**

Panel to access the **Control Panel** window and then click the **Users** icon. To change a password, right-click user record and select **Set Password**.

3.3 Discover the network

If you enabled automatic network discovery, wait for initial network discovery completion. This process can take time, depending on size and complexity of your network. For large networks, we recommend that you let NetXMS run over night to gather the majority of network information available. You can watch discovery process in real time using NetXMS Management Console. Go to Object Browser or open default network map and see for new devices and networks.

Please note that for successful network discovery, your network should meet the following requirements:

- NetXMS server should have access to switches and routers via SNMP.
- All your network devices should use the same community string, and this community string should be specified as value for *DefaultCommunityString* parameter in server's configuration.

3.4 Manually add nodes

If the automatic network discovery does not detect all of your hosts or devices, or you decide not to use network discovery at all, you may need to manually add monitored nodes to the system. The easiest way to accomplish this is to click **Add Node** on the **Tools** menu. You will be prompted for new node name and IP address. Please note that adding new node object may take some time, especially if a node is down or behind a firewall. After successful creation, new node object will be placed into appropriate subnets automatically.

As soon as you add a new node to the system, NetXMS server will start regular polling to determine node status.

4 Objects

4.1 Overview

All network infrastructure monitored by NetXMS inside monitoring system represented as a set of *objects*. Each object represents one physical or logical entity (like host or network interface), or group of them. Objects organized into hierarchical structure. There are 12 different object classes:

Entire Network	Abstract object representing root of IP topology tree. All subnet objects located under it. System can have only one object of this class.
Subnet	Object representing IP subnet. Typically objects of this class created automatically by the system to reflect system's knowledge of IP topology.
Node	Object representing physical host or network device. These objects can be created either manually by administrator or automatically during network discovery process.
Cluster	Object representing cluster consisted of two or more hosts.
Interface	Object representing network interface of node. These objects created automatically by the system during configuration polls.
Network Service	Object representing network service running on a node (like http or ssh).
VPN Connector	Object representing VPN tunnel endpoint. Such objects can be created to add VPN tunnels to network topology known y NetXMS server.
Service Root	Abstract object representing root of your service tree. System can have only one object of this class.
Container	Grouping object which can contain nodes, subnets, clusters, conditions, or other containers. With help of container objects you can build object's tree which represents logical hierarchy of IT services in your organization.
Condition	Object representing complicated condition – like "cpu on node1 is overloaded and node2 is down for more than 10 minutes".
Template Root	Abstract object representing root of your template tree.
Template Group	Grouping object which can contain templates or other template groups.
Template	Data collection template. See <i>Data Collection</i> section for more information about templates.

Every object has set of attributes; some of them are common (like id and name), while other depends on object class – for example, attribute "SNMP community string" have only node objects.

4.2 Top Level Objects

NetXMS has three top level objects – **Entire Network**, **Service Root**, and **Template Root**. These objects served as an abstract root for appropriate object tree. The following table represents possible child object classes for top level objects:

Object	Possible childs
Entire Network	Subnet
Service Root	Container Subnet Node Cluster

	Condition
Template Root	Template Group Template

All top level objects has only one editable attribute – object's name.

4.3 Subnet objects

Subnet objects represents IP subnets. These objects created automatically by NetXMS server to represent known IP topology. Subnet objects can contain only node objects, which are placed automatically inside appropriate subnet object based on interface configuration. Subnet objects has only one editable attribute – object's name.

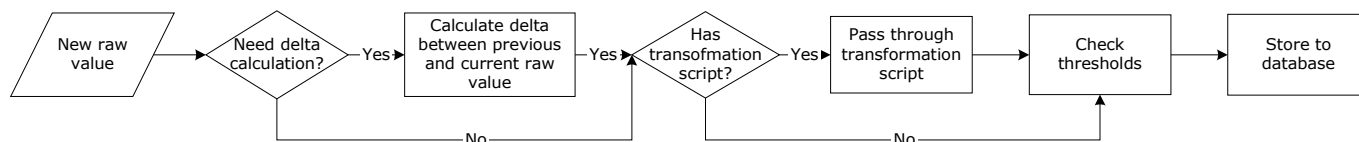
5 Data Collection

5.1 How data collection works

Every node can have many data collection items configured (see [Basic Concepts](#) chapter for detailed description of DCI). NetXMS server has a set of threads dedicated to data collection, called *Data Collectors*, used to gather information from the nodes according to DCI configuration. You can control how many data collectors will run simultaneously, by changing server configuration parameter *NumberOfDataCollectors*.

All configured DCIs are checked for polling requirement every two seconds and if DCI needs to be polled, appropriate polling request is placed into internal data polling queue. First available data collector will pick up the request and gather information from the node according to DCI configuration. If a new value was received successfully, it's being stored in the database, and thresholds are checked. After threshold checking, data collector is ready for processing new request. Processing of a newly received parameter value is outlined on Figure 3

Figure 3: Newly received parameter processing



5.2 DCI configuration

5.2.1 Basic configuration

Data collection for a node can be configured using management console. To open data collection configuration window, right-click node object in object browser or on a map, and click **Data Collection**. You will see the list of configured data collection items. From here, you can add new or change existing parameters to monitor. Usual way to do something with DCIs is to right-click on appropriate record in the list and select a required action from popup menu.

When you create new DCI or open an existing one, you will see a lot of attributes. The list of definitions and descriptions for the attributes is given below.

5.2.1.1 Description

Description is a free-form text string describing DCI. It is not used by the server and is intended for better information understanding by operators. If you use the **Select** button to select a parameter from the list, description field will be filled automatically.

5.2.1.2 Parameter

Name of the parameter of interest, used for making a request to target node. For NetXMS agent and internal parameters it will be parameter name, and for SNMP agent it will be an SNMP OID. You can use the **Select** button for easier selection of required parameter name.

5.2.1.3 Origin

Origin of data (or method of obtaining data). Possible origins are NetXMS agent, SNMP agent, CheckPoint SNMP agent, Internal (data generated inside NetXMS server process), or Push Agent. Last origin is very different from all others, because it represents DCIs whose values are pushed to server by external program (usually via **npxpush** utility) instead of being polled by the server based on the schedule.

5.2.1.4 Data Type

Data type for the parameter. Can be one of the following: Integer, Unsigned Integer, 64-bit Integer, 64-bit Unsigned Integer, Float (floating point number), or String. Selected data type affects collected data processing — for example, you cannot use operations like “less than” or “greater than” on strings. If you select parameter from the list using the **Select** button, correct data type will be set automatically.

5.2.1.5 Polling Interval

An interval between consecutive polls, in seconds. If you select the **Use advanced scheduling** option, this field has no meaning and will be disabled.

5.2.1.6 Use Advanced Schedule

If you turn on this flag, NetXMS server will use custom schedule for collecting DCI values instead of fixed intervals. This schedule can be configured on the **Schedule** page. Advanced schedule consists of one or more records; each representing desired data collection time in cron-style format. Record has five fields, separated by spaces: minute, hour, day of month, month, and day of week. Allowed values for each field are:

Table 2: Advanced Schedule field values

Field	Value
minute	0 — 59
hour	0 — 23
day of month	1 — 32
month	0 — 12
day of week	0 — 7 (0 or 7 is Sunday)

A field may be an asterisk (*), which always stands for “any”.

Some examples:

```
5 0 * * *
```

Run five minutes after midnight, every day.

```
15 14 1 * *
```

Run at 14:15 on the first day of every month.

5.2.1.7 Associate with cluster resource

In this field you can specify cluster resource associated with DCI. Data collection and processing will occur only if node you configured DCI for is current owner of this resource. This field is valid only for cluster member nodes.

5.2.1.8 Retention Time

This attribute specifies how long the collected data should be kept in database, in days. Minimum retention time is 1 day and maximum is not limited. However, keeping too many collected values for too long will lead to significant increase of your database size and possible performance degradation.

5.2.1.9 Status

DCI status can be one of the following: *Active*, *Disabled*, *Not Supported*. Server will collect data only if the status is *Active*. If you wish to stop data collection without removing DCI configuration and collected data, the *Disabled* status can be set manually. If requested parameter is not supported by target node, the *Not Supported* status is set by the server.

5.2.2 Data Transformations

In simplest case, NetXMS server collects values of specified parameters and stores them in the database. However, you can also specify various transformations for original value. For example, you may be interested in a delta value, not in a raw value of some parameter. Or, you may want to have parameter value converted from bytes to kilobytes. All transformations will take place after receiving new value and before threshold processing.

Data transformation consists of two steps. On the first step, delta calculation is performed. You can choose four types of delta calculation:

Table 3: Delta calculation types

Calculation Type	Description
<i>None</i>	No delta calculation performed. This is the default setting for newly created DCI.
<i>Simple</i>	Resulting value will be calculated as a difference between current raw value and previous raw value. By raw value is meant the parameter value originally received from host.
<i>Average per second</i>	Resulting value will be calculated as a difference between current raw value and previous raw value, divided by number of seconds passed between current and previous polls.
<i>Average per minute</i>	Resulting value will be calculated as a difference between current raw value and previous raw value, divided by number of minutes passed between current and previous polls.

On the second step, custom transformation script is executed (if presented). By default, newly created DCI does not have a transformation script. If transformation script is presented, the resulting value of the first step is passed to the transformation script as a parameter; and a result of script execution is a final DCI value. For more information about NetXMS scripting language, please consult [NetXMS Scripting Language](#) chapter in this manual.

5.2.3 Thresholds

5.2.3.1 Overview

For every DCI you can define one or more thresholds. Each threshold there is a pair of condition and event – if condition becomes true, associated event is generated. To configure thresholds, open the data collection editor for node or template, right-click on the DCI record and select **Edit** from the popup menu, then select the **Thresholds** page. You can add, modify and delete thresholds using buttons below the threshold list. If you need to change the threshold order, select one threshold and use arrow buttons located on the right to move the selected threshold up or down.

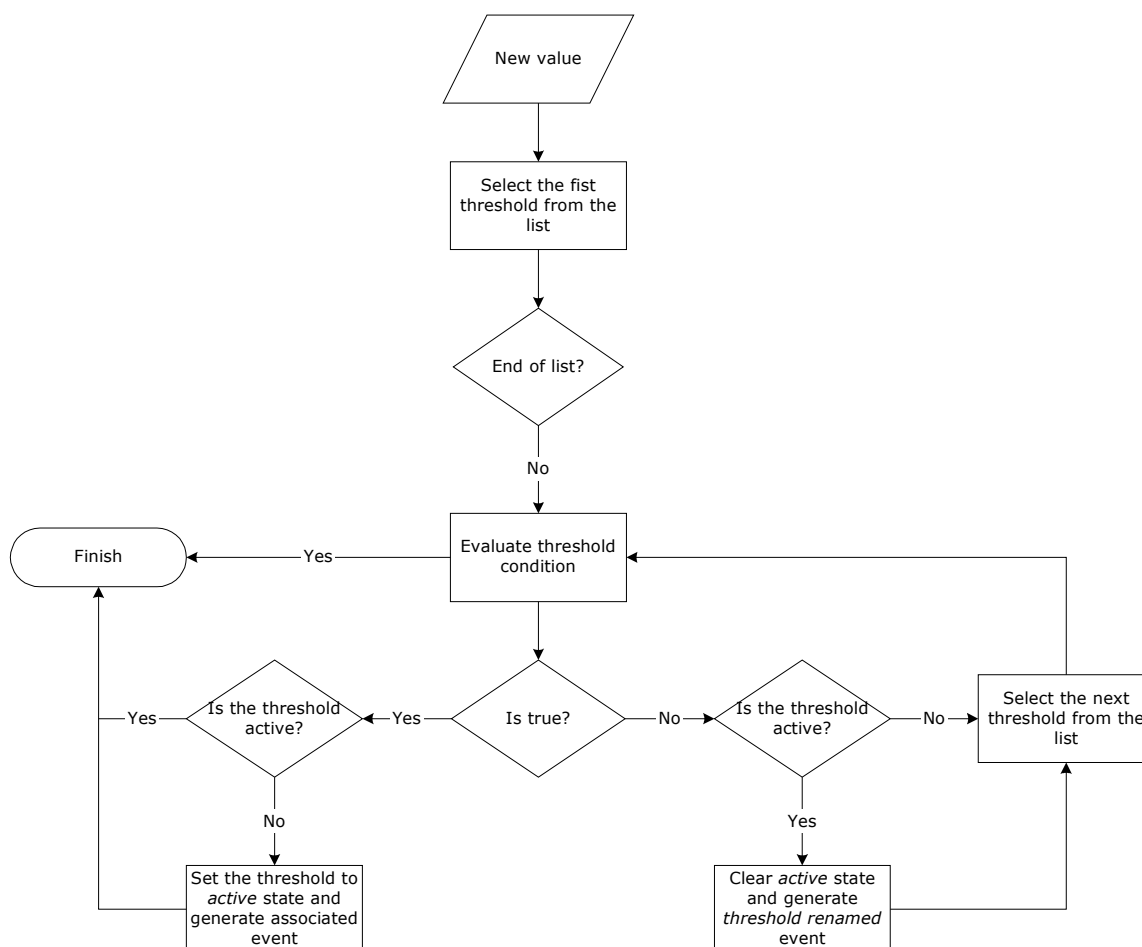
5.2.3.2 Instance

Each DCI has an *Instance* attribute, which is a free-form text string, passed as a 6th parameter to events associated with thresholds. You can use this parameter to distinguish between similar events related to different instances of the same entity – for example, if you have an event generated when file system was low on free space, you can set the *Instance* attribute to file system mount point.

5.2.3.3 Threshold Processing

Threshold processing algorithm is outlined on Figure 4.

Figure 4: Threshold processing algorithm



As you can see from this flowchart, threshold order is very important. Let's consider the following example: you have DCI representing CPU utilization on the node, and you wish two different events to be generated – one when CPU utilization exceeds 50%, and another one when it exceeds 90%. What happens when you place threshold "> 50" first, and "> 90" second? Table 4

shows values received from host and actions taken by monitoring system (assuming that all thresholds initially unarmed):

Table 4: Actions taken by monitoring system

Value	Action
10	Nothing will happen.
55	When checking first threshold ("> 50"), the system will find that it's not active, but condition evaluates to true. So, the system will set threshold state to "active" and generate event associated with it.
70	When checking first threshold ("> 50"), the system will find that it's already active, and condition evaluates to true. So, the system will stop threshold checking and will not take any actions.
95	When checking first threshold ("> 50"), the system will find that it's already active, and condition evaluates to true. So, the system will stop threshold checking and will not take any actions.

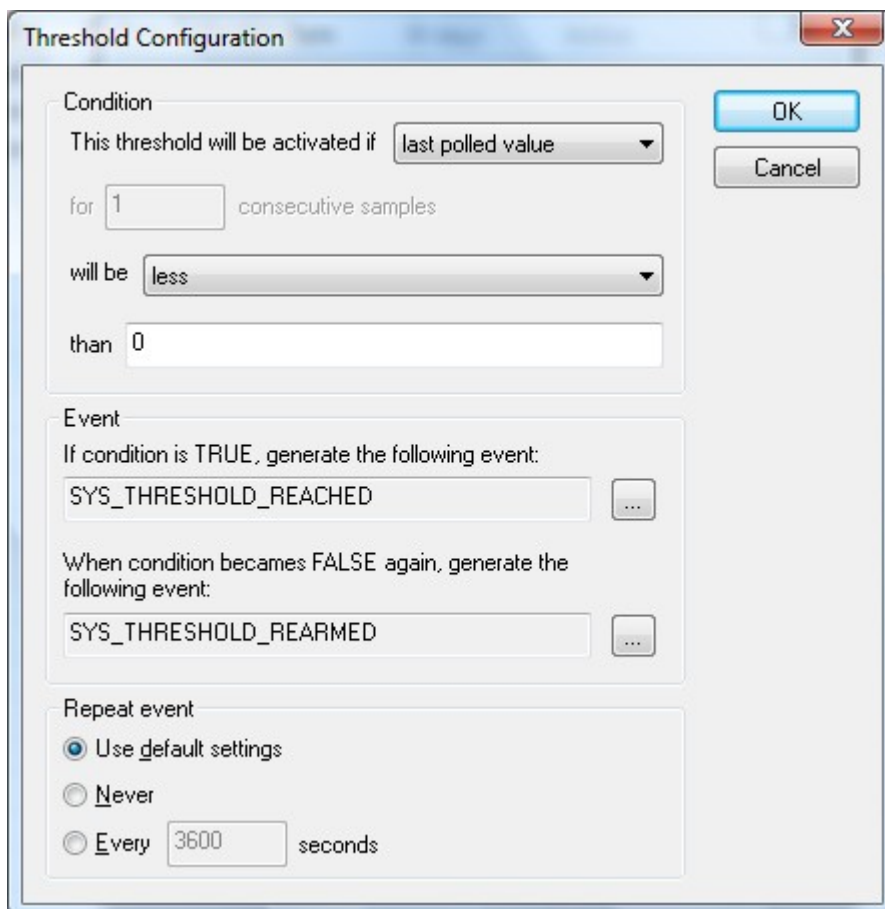
Please note that second threshold actually is not working, because it's "masked" by the first threshold. To achieve desired results, you should place threshold "> 90" first, and threshold "> 50" second.

You can disable threshold ordering by checking **Always process all thresholds** checkbox. If it is marked, system will always process all thresholds.

5.2.3.4 Threshold Configuration

When adding or modifying a threshold, you will see the following dialog:

Figure 5: Threshold Configuration window



The image shows a 'Threshold Configuration' dialog box with the following sections:

- Condition:** A group box containing:
 - 'This threshold will be activated if' followed by a dropdown menu showing 'last polled value'.
 - 'for' followed by a text input field containing '1' and the text 'consecutive samples'.
 - 'will be' followed by a dropdown menu showing 'less'.
 - 'than' followed by a text input field containing '0'.
- Event:** A group box containing:
 - 'If condition is TRUE, generate the following event:' followed by a text input field containing 'SYS_THRESHOLD_REACHED' and a button with three dots.
 - 'When condition becomes FALSE again, generate the following event:' followed by a text input field containing 'SYS_THRESHOLD_REARMED' and a button with three dots.
- Repeat event:** A group box containing:
 - Three radio buttons: 'Use default settings' (which is selected), 'Never', and 'Every'.
 - Next to the 'Every' radio button is a text input field containing '3600' followed by the text 'seconds'.

On the right side of the dialog are 'OK' and 'Cancel' buttons.

First, you have to select what value will be checked:

- *last polled value*
Last value will be used.
- *average value*
An average value for last *n* polls will be used (you have to configure a desired number of polls).
- *mean deviation*
A mean absolute deviation for last *n* polls will be used (you have to configure a desired number of polls). Additional information on how mean absolute deviation calculated can be found here: http://en.wikipedia.org/wiki/Mean_deviation.
- *diff with previous value*
A delta between last and previous values will be used. If DCI data type is string, system will use 0, if last and previous values match; and 1, if they don't.
- *data collection error*
An indicator of data collection error. Instead of DCI's value, system will use 0 if data collection was successful, and 1 if there was a data collection error. You can use this type of thresholds to catch situations when DCI's value cannot be retrieved from agent.

Second, you have to select comparison function. Please note that not all functions can be used for all data types. Below is a compatibility table:

Table 5: Functions compatibility table

	Integer	Unsigned Integer	Int64	Unsigned Int64	Float	String
less	X	X	X	X	X	
less or equal	X	X	X	X	X	
equal	X	X	X	X	X	X
greater or equal	X	X	X	X	X	
greater	X	X	X	X	X	
not equal	X	X	X	X	X	X
like						X
not like						X

Third, you have to set a value to check against. If you use *like* or *not like* functions, value is a pattern string where you can use metacharacters: "*" (asterisk), which means "any number of any characters", and "?" (question mark), which means "any character".

Fourth, you have to select events to be generated when the condition becomes true or returns to false. By default, system uses SYS_THRESHOLD_REACHED and SYS_THRESHOLD_REARMED events, but in most cases you will change it to your custom events.

You can also configure threshold to resend activation event if threshold's condition remain true for specific period of time. You have three options – *default*, which will use server-wide settings, *never*, which will disable resending of events, or specify interval in seconds between repeated events.

5.2.3.5 Thresholds and Events

You can choose any event to be generated when threshold becomes active or returns to inactive state. However, you should avoid using predefined system events (their names usually start with SYS_ or SNMP_). For example, you set event SYS_NODE_CRITICAL to be generated when CPU utilization exceeds 80%. System will generate this event, but it will also generate the same event when node status will change to CRITICAL. In your event processing configuration, you will be unable to determine actual reason for that event generation, and probably will get some unexpected results. If you need custom processing for specific threshold, you should create your

own event first, and use this event in the threshold configuration. NetXMS has some preconfigured events that are intended to be used with thresholds. Their names start with DC_.

The system will pass the following six parameters to all events generated as a reaction to threshold violation:

1. Parameter name (DCI's *name* attribute)
2. DCI description
3. Threshold value
4. Actual value
5. Unique DCI identifier
6. Instance (DCI's *instance* attribute)

For example, if you are creating a custom event that is intended to be generated when file system is low on free space, and wish to include file system name, actual free space, and threshold's value into event's message text, you can use message template like this:

"File system %6 has only %4 bytes of free space (threshold: %3 bytes)".

For events generated on threshold's return to inactive state (default event is SYS_THRESHOLD_REARMED), parameter list is different:

1. Parameter name (DCI's *name* attribute)
2. DCI description
3. Unique DCI identifier
4. Instance (DCI's *instance* attribute)

5.2.4 Push parameters

NetXMS gives you ability to push DCI values when you need it instead of polling them on specific time intervals. To be able to push data to the server, you should take the following steps:

1. Set your DCI's origin to *Push Agent* and configure other properties as usual, excluding polling interval which is meaningless in case of pushed data.
2. Create separate user account or pick an existing one and give "Push Data" right on the DCI owning node to that user.
3. Use **npxpush** utility or client API for pushing data.

5.3 Templates

5.3.1 What is templates

Often you have a situation when you need to collect same parameters from different nodes. Such configuration making may easily fall into repeating one action many times. Things may become even worse when you need to change something in already configured DCIs on all nodes – for example, increase threshold for CPU utilization. To avoid these problems, you can use *data collection templates*. Data collection template (or just template for short) is a special object, which can have configured DCIs similar to nodes. When you create template and configure DCIs for it, nothing happens – no data collection will occur. Then, you can *apply* this template to one or multiple nodes – and as soon as you do this, all DCIs configured in the template object will appear in the target node objects, and server will start data collection for these DCIs. If you then change something in the template data collection settings – add new DCI, change DCI's configuration, or remove DCI – all changes will be reflected immediately in all nodes associated with the template. You can also choose to remove template from a node. In this case, you will have two options to deal with DCIs configured on the node through the template – remove all such DCIs or leave them, but remove relation to the template. If you delete template object itself, all DCIs created on nodes from this template will be deleted as well.

Please note that you can apply an unlimited number of templates to a node – so you can create individual templates for each group of parameters (for example, generic performance parameters, MySQL parameters, network counters, etc.) and combine them, as you need.

5.3.2 Creating template

To create a template, right-click on **Template Root** or **Template Group** object in the Object Browser, and click **Create** and then select **Template**. Enter a name for a new template and click **OK**.

5.3.3 Configuring templates

To configure DCIs in the template, right-click on **Template** object in the Object Browser, and select **Data Collection** from the popup menu. **Data collection editor** window will open. Now you can configure DCIs in the same way as the node objects.

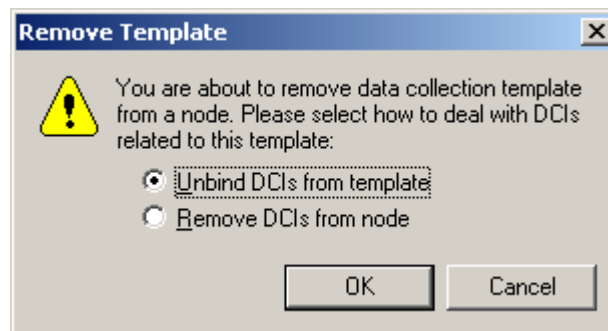
5.3.4 Applying template to node

To apply a template to one or more nodes, right-click on template object in Object Browser and select **Apply** from popup menu. Node selection dialog will open. Select the nodes that you wish to apply template to, and click **OK** (you can select multiple nodes in the list by holding CTRL key). Please note that if data collection editor is open for any of the target nodes, either by you or another administrator, template applying will be delayed until data collection editor for that node will be closed.

5.3.5 Removing template from node

To remove a link between template and node, right-click on **Template** object in the Object Browser and select **Unbind** from popup menu. Node selection dialog will open. Select one or more nodes you wish to unbind from template, and click **OK**. The system will ask you how to deal with DCIs configured on node and associated with template:

Figure 6: Remove Template window



If you select **Unbind DCIs from template**, all DCIs related to template will remain configured on a node, but association between the DCIs and template will be removed. Any further changes to the template will not be reflected in these DCIs. If you later reapply the template to the node, you will have two copies of each DCI – one standalone (remaining from unbind operation) and one related to template (from new apply operation). Selecting **Remove DCIs from node** will remove all DCIs associated with the template. After you click OK, node will be unbound from template.

5.3.6 Macros in template items

You can use various macros in *name*, *description*, and *instance* fields of template DCI. These macros will be expanded when template applies to node. Macro started with %{ character combination and ends with } character. The following macros are currently available:

Macro	Expands to...
node_id	Node unique id
node_name	Node name
node_primary_ip	Node primary IP address
script:name	String returned by script <i>name</i> . Script should be stored in script library (accessible via Control Panel -> Script Library). Inside the script, you can access current node's properties via \$node variable.

For example, if you wish to insert node's IP address into DCI description, you can enter the following in the description field of template DCI:

```
My ip address is %{node_primary_ip}
```

When applying to node with primary IP address 10.0.0.1, on the node will be created DCI with the following description:

```
My ip address is 10.0.0.1
```

Please note that if you change something in the node, name for example, these changes will not be reflected automatically in DCI texts generated from macros. However, they will be updated if you reapply template to the node.

5.4 Working with collected data

Once you setup DCI, data starts collecting in the database. You can access this data and work with it in different ways.

5.4.1 View collected data in graphical form

You can view collected data in a graphical form, as a line chart. To view values of some DCI as a chart, first open either Data Collection Editor or Last Values view for a host. You can do it from the Object Browser or map by selection host, right-clicking on it, and selecting **Data collection** or **Last DCI values**. Then, select one or more DCIs (you can put up to 16 DCIs on one graph), right-click on them and choose **Graph** from the pop-up menu. You will see graphical representation of DCI values for the last hour.

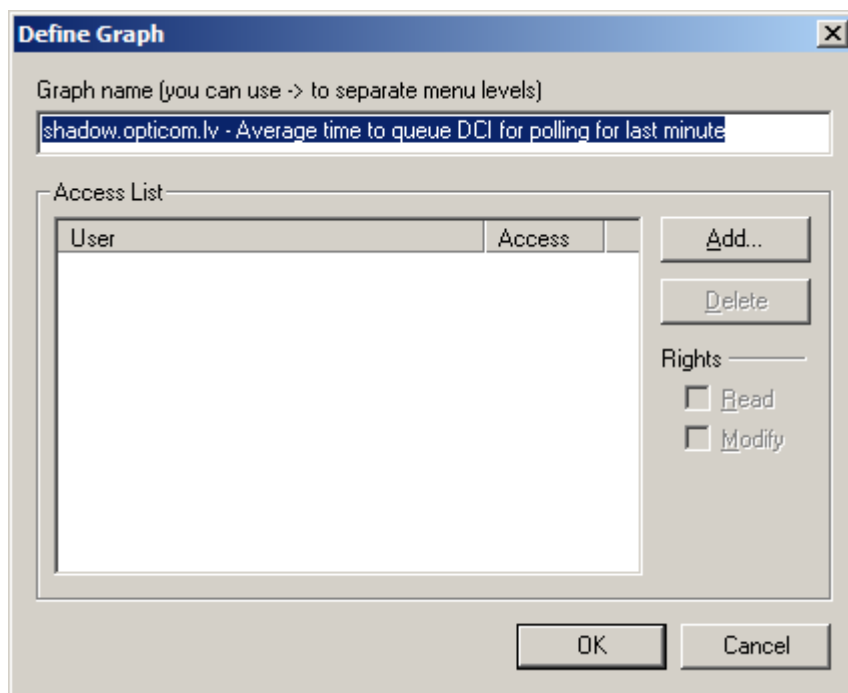
When the graph is open, you can do various tasks:

- Select different time interval
By default, you will see data for the last hour. You can select different time interval in two ways:
 1. Select new time interval from presets, by right-clicking on the graph, and then selecting **Presets** and appropriate time interval from the pop-up menu.
 2. Set time interval in graph properties dialog. To access graph properties, right-click on the graph, and then select **Properties** from the pop-up menu. Alternatively, you can select **Properties** on the **Graph** menu (main application menu). In the properties dialog, you will have two options: select exact time interval (like 12/10/2005 from 10:00 to 14:00) or select time interval based on current time (like last two hours).
- Turn on automatic refresh
You can turn on automatic graph refresh at a given interval in graph properties dialog. To access graph properties, right-click on it, and select **Properties** from the pop-up menu. Alternatively, you can select **Properties** on the **Graph** menu (main application menu). In the properties dialog, select the **Refresh automatically** checkbox and enter a desired refresh interval in seconds in edit box below. When automatic refresh is on, you will see "Autoupdate" message in the status bar of graph window.
- Change colors
You can change colors used to paint lines and graph elements in the graph properties dialog. To access graph properties, right-click on it, and select **Properties** from the pop-up menu. Alternatively, you can select **Properties** on the **Graph** menu (main application

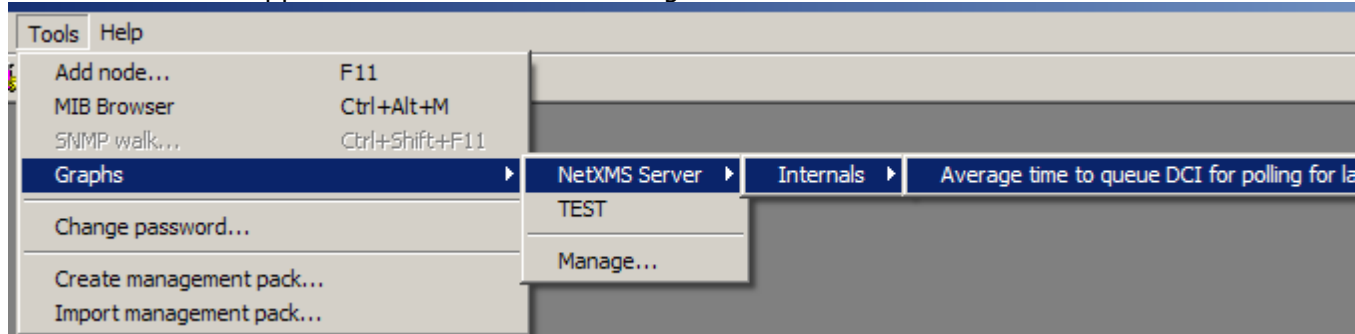
menu). In the properties dialog, click on colored box for appropriate element to choose different color.

- Save current settings as predefined graph

You can save current graph settings as predefined graph to allow quick and easy access in the future to information presented on graph. Preconfigured graphs can be used either by you or by other NetXMS users, depending on settings. To save current graph configuration as predefined graph, select **Save as predefined** from **Graph** menu. The following dialog will appear:



In **Graph name** field, enter desired name for your predefined graph. It will appear in **Tools -> Graphs** menu in console exactly as written here. You can use & character to set shortcut key for menu item, and -> character pair to create submenus. For example, if you name your graph *NetXMS Server->Internals->Average time to queue DCI for polling for last minute* it will appear in the menu as following:



In the **Access List** area you can add users who will have access to this graph. Note that user creating the graph will always have full access to it, even if he is not in access list. If you need to change or delete predefined graph, you can do it via **Tools -> Graphs -> Manage** menu.

5.4.2 View collected data in textual form

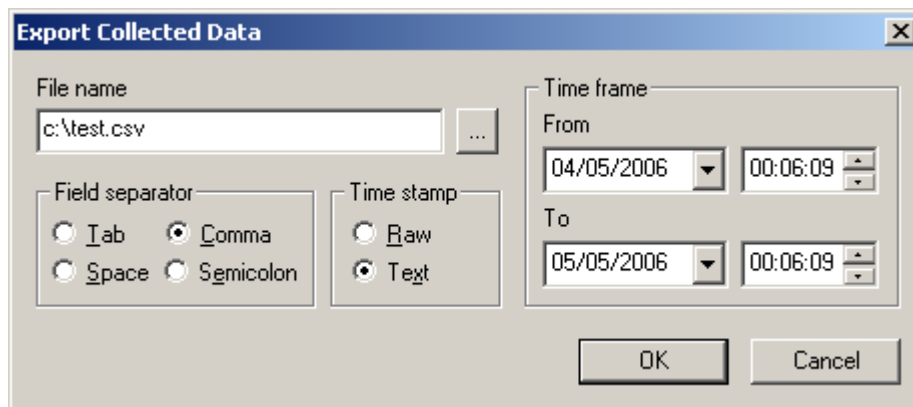
You can view collected data in a textual form, as a table with two columns – timestamp and value. To view values of some DCI as a table, first open either **Data Collection Editor** or **Last Values**

view for a host. You can do it from the Object Browser or map by selection host, right-clicking on it, and selecting **Data collection** or **Last DCI values**. Then, select one or more DCIs (each DCI data will be shown in separate window), right-click on them and choose **Show history** from the pop-up menu. You will see the last 1000 values of the DCI.

5.4.3 Export DCI data

You can export collected data to a text file. To export DCI data, first open either **Data Collection Editor** or **Last Values view** for a host. You can do it from the Object Browser or map by selection host, right-clicking on it, and selecting **Data collection** or **Last DCI values**. Then, select one DCI, right-click on it and select **Export data** from the pop-up menu. Export configuration dialog will appear (Figure 7).

Figure 7: Export configuration dialog



Enter the name of the file where you wish to save the data. You can use the ... button next to the **File name** box, to open the file system browser. Then, select separator to be used between timestamps and values, time stamp format and time frame. Time stamp format can be either *Raw* – number representing time in a UNIX timestamp format, or *Text* – string in format "dd/mm/yyyy HH:MM:SS".

Finally, click **OK** and wait for export process to complete.

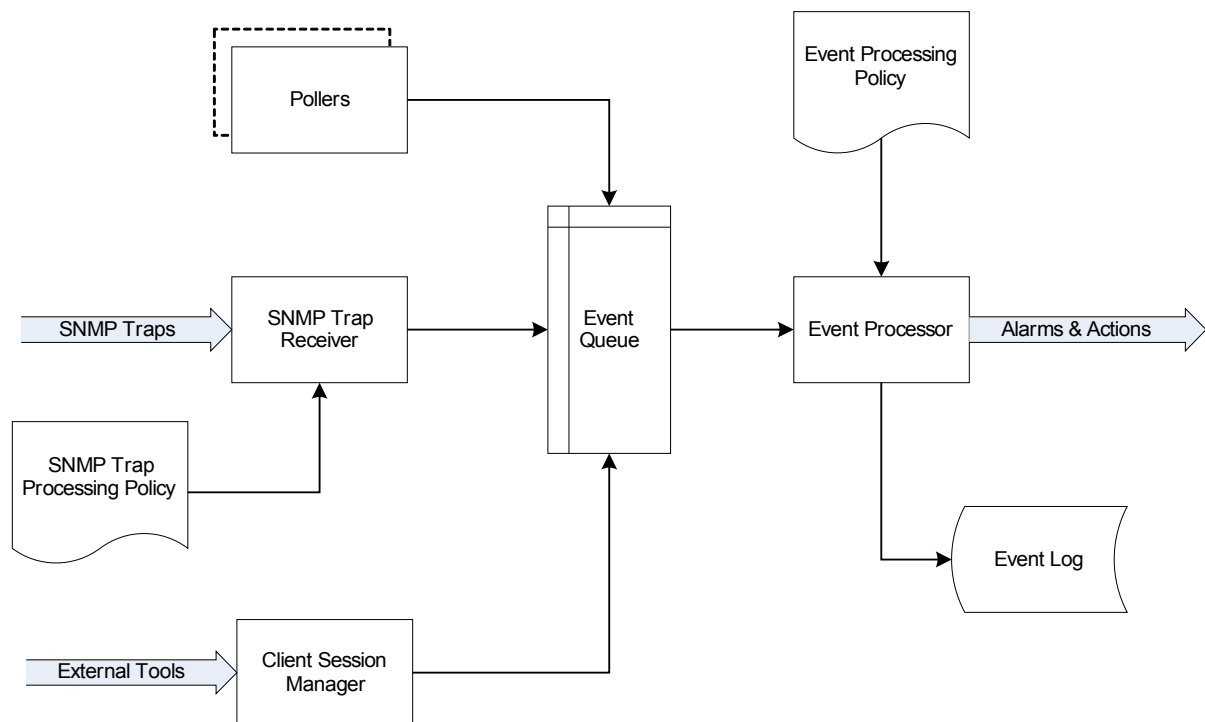
6 Event Processing

Event processing is one of core components of NetXMS. It determines how the monitoring system will react to various events.

6.1 Event Processing Overview

The following flowchart outlines event flow inside the monitoring system:

Figure 8: Event flow inside the monitoring system



As you can see on the flowchart, events can come from various sources: polling processes (status, configuration, discovery, and data collection), SNMP traps, and directly from external applications via client library. All incoming events go to single event queue for processing. A special process, called *Event Processor*, takes events from the queue one by one and matches them against Event Processing Policy. As a result, alarms may be generated and actions may be executed. If event has *write to log* attribute set, it is written to NetXMS event log at the end of processing. Although it may seem that processing all events one by one may become a bottleneck in the system, this should not be the case. Event processor is highly optimized, and all potentially long operations (like action execution) are performed by separate processes.

6.2 Event Processing Policy

Actions taken by event processor for any specific event determined by set of rules called *Event Processing Policy*. Every rule has two parts – matching part, which determines if rule is appropriate for current event, and action part, which determines actions to be taken for matched events. Matching part consists of four fields:

- Source** Event's source node. This field can be set to *any*, which matches any node, or contain a list of nodes, subnets, or containers. If you specify subnet or container, any node within it will be matched.
- Event** Event code. This field can be set to *any*, which matches any event, or list of event

codes.

Severity Event's severity. This field contains selection of event severities to be matched.

Script Optional matching script written in NXSL. If this field is empty, no additional checks performed. Otherwise, event will be considered as matched only if script will return non-zero (TRUE) return code. For more information about NetXMS scripting language, please consult [NetXMS Scripting Language](#) chapter in this manual.

In action part you can set alarm generation, situation update, and list of actions to be executed. Every rule can also have a free-form textual comment.

Each event passes through all rules in the policy, so if it matches to more than one rule, actions specified in all matched rules will be executed. You can change this behavior by setting *Stop Processing* flag for the rule. If this flag is set and rule matched, processing of current event will be stopped.

You can create and modify Event Processing Policy using **Event Processing Policy Editor**. To access the **Event Processing Policy Editor** window, press F9 or on the **View** menu click **Control Panel** to access the **Control Panel** window and then click the **Event Processing Policy** icon.

Examples:

3	IPSO	* Any	Major Critical	None	Mail.Dmitry (GSM) Mail.Leonid (GSM)
---	------	-------	-------------------	------	--

This rule defines that for every major or critical event originated from any node within "IPSO" container two e-mail actions should be executed.

28	* Any	NOKIA_CFG_CHANGED NOKIA_CFG_SAVED NOKIA_LOW_DISK_SPACE NOKIA_NO_DISK_SPACE	* Any	?%m	None
----	-------	---	-------	-----	------

This rule defines that for events NOKIA_CFG_CHANGED, NOKIA_CFG_SAVED, NOKIA_LOW_DISK_SPACE, and NOKIA_NO_DISK_SPACE, originated from any node, system should generate alarm with text "%m" (which means "use event's message text") and severity equal to event's severity.

6.3 Alarms

6.3.1 Alarms Overview

As a result of event processing some events can be shown up as *alarms*. Usually alarm represents something that needs attention of network administrators or network control center operators, for example low free disk space on a server. Every alarm has the following attributes:

Creation time	Time when alarm was created.
Last change time	Time when alarm was last changed (for example, acknowledged).
State	Alarm can be in one of three states: <div> <div>Outstanding</div> <div>Acknowledged</div> <div>Terminated</div> </div> <div> <div>New alarm;</div> <div>When network administrator sees an alarm, he may <i>acknowledge</i> it to indicate that somebody already aware of that problem and working on it;</div> <div>Inactive alarm. When problem is solved, network administrator can terminate alarm – this will remove alarm from active alarms list and it will not be seen in console, but alarm record will remain in database.</div> </div>

Message	Message text (usually derived from originating event's message text).
Severity	Alarm's severity – Normal, Warning, Minor, Major, or Critical.
Source	Source node (derived from originating event).
Key	Text string used to identify duplicate alarms and for automatic alarm termination.

6.3.2 Generating Alarms

To generate alarms from events, you should edit "Alarm" field in appropriate rule of Event Processing Policy. Alarm configuration dialog will look like this:

Figure 9: Alarm configuration dialog

You should select **Generate new alarm** radio button to enable alarm generation from current rule. In the **Message** field enter alarm's text, and in the alarm key enter value which will be used for repeated alarms detection and automatic alarm termination. In both fields you can use macros described in the [Macros for Event Processing](#) chapter.

You can also configure sending of additional event if alarm will stay in **Outstanding** state for given period of time. To enable this, enter desired number of seconds in **Seconds** field, and select event to be sent. Entering value of 0 for seconds will disable additional event sending.

6.3.3 Automatic Alarm Termination

You can terminate all active alarms with given key as a reaction for the event. To do this, select **Terminate alarm** radio button in alarm configuration dialog and enter value for alarm key. For that field you can use macros described in the [Macros for Event Processing](#) chapter.

6.4 Situations

6.4.1 Situations Overview

Situations is a special type of event processing objects allowing you to track current state of your infrastructure and process events according to it. Each situation has one or more instances, and each instance has one or more attributes. Situation objects allows you to store information about current situation in attributes and then use this information in event processing. For example, if you have one service (service A) depending on another (service B), and in case of service B failure you wish to get alarm about service B failure, and not about consequent service A failure. To accomplish this, you can do the following:

1. Create situation object named "ServiceStatus";
2. In event processing policy, for processing of event indicating service B failure, add situation attribute update: update situation "ServiceStatus", instance "Service_B", set attribute "status" to "failed";
3. In event processing policy, for rule generating alarm in case of service A failure, add additional filtering using script – to match this rule only if service B is not failed. Your script may look like following:

```
sub main()
{
    s = FindSituation("ServiceStatus", "Service_B");
    if (s != NULL)
    {
        if (s->status == "failed")
            return 0;    // Don't match rule
    }
    return 1;    // Match rule
}
```

7 Troubleshooting

In this chapter, you will find most common problems, which can be encountered by NetXMS user or administrator, and recommended solutions for them.

7.1 Server problems

Problem: server doesn't start at all or exit immediately after startup.

Table 6: Server problems

Possible cause	Solution
Server's configuration file is missing or incorrect	Make sure that file <code>netxmsd.conf</code> is located in the correct directory. Run <code>netxmsd -C</code> command to test configuration file.
Server cannot connect to the database	Check that all database credentials are set correctly and that database server is up. Run <code>netxmsd --debug=7</code> to get extended diagnostic messages.
Database is locked or corrupted	This may happen after server crash or incorrect shutdown. Make sure that no other instances on <code>netxmsd</code> are running, and then run <code>nxdbmgr check</code> command. You will be prompted to unlock the database. Answer "Y" and wait for database checking process to complete, and then try to start <code>netxmsd</code> again.

7.2 Agent problems

Problem: agent doesn't start.

Table 7: Agent problems

Possible cause	Solution
Agent's configuration file is missing or incorrect	Make sure that file <code>nxagentd.conf</code> is located in the correct directory. Run <code>nxagentd -C</code> command to test configuration file.
Required DLLs are missing	Run <code>nxagentd -D</code> to check if process starts or not and to get extended diagnostic messages. If process doesn't start at all, try reinstalling the agent.
TCP listen port used by agent (4700 by default) already used by other application	Change agent's listen port by setting <code>ListenPort</code> parameter in <code>nxagentd.conf</code> or stop application which currently uses this port.

8 Complete Reference

8.1 Server Configuration

8.1.1 File: netxmsd.conf

File netxmsd.conf is a master configuration file for NetXMS server. It contains only information necessary for establishing database connection. Default location for this file is `/etc/netxmsd.conf` on UNIX systems and `C:\netxmsd.conf` on Windows.

The file can contain the following parameters in Parameter = Value format:

Table 8: netxmsd.conf parameters

Parameter	Description	Default
CodePage	Code page used by NetXMS server. Has no effect on Windows or if server was compiled without iconv support.	Depends on your system, usually ISO8859-1
CreateCrashDumps	Control creation of server's crash dumps. Possible values: <i>yes</i> or <i>no</i> . Has effect only on Windows platforms.	<i>No</i>
DBDriver	Database driver to be used	No default value
DBDrvParams	Additional driver-specific parameters	Empty string
DBLogin	Database user name	<i>netxms</i>
DBName	Database name (not used by ODBC driver)	<i>netxms_db</i>
DBPassword	Database user's password	Empty password
DBServer	Database server (ODBC source name for ODBC driver)	<i>localhost</i>
LogFailedSQLQueries	Control logging of failed SQL queries. Possible values: <i>yes</i> or <i>no</i> .	<i>yes</i>
LogFile	Server's log file. To write log to syslog (or Event Log on Windows), use {syslog} as file name.	{syslog}

Configuration file example:

```
#
# Sample configuration file for NetXMS server
#

DBDriver = mysql.ldr
DBServer = localhost
DBName = netxms_db
DBLogin = netxms
DBPassword = password
LogFailedSQLQueries = yes
LogFile = {syslog}
```

8.1.2 Table in Database: config

After reading the connection information from the configuration file, the server is connected to the database and operational. Additional configuration is stored in the database itself and accessible through the **Server Configuration** window in the console. To access the **Server Configuration** window, press F9 or on the **View** menu click **Control Panel** to access the **Control Panel** window and then click the **Server Configuration** icon.

To edit a setting, double click on the row in the table or right-click and select **Edit**. To add a new configuration item-value pair, right-click on the table and select **New**. To delete a pair, select it, right-click and select **Delete**.

Please note that changes to most of the settings will take effect only after server restart.

Useful settings include:

Table 9: Server configuration parameters

Parameter	Description	Default
ActiveDiscoveryInterval	Interval in seconds between active network discovery polls.	7200
ActiveNetworkDiscovery	Enable (1) or disable (0) active network discovery.	0
AgentCommandTimeout	Timeout in milliseconds for commands sent to agent. If agent did not respond to command within given number of seconds, command considered as failed.	2000
AgentUpgradeWaitTime	Maximum wait time in seconds for agent restart after upgrade. If agent cannot be contacted after this time period, upgrade process is considered as failed.	600
AllowDirectSMS	Allow (1) or disallow (0) sending of SMS via NetXMS server using nxsms utility.	0
AllowedCiphers	Control what ciphers server can use for connection encryption. Value for this parameter is a cipher code. To enable more than one cipher, the codes should be summed up. Possible cipher codes: <div style="margin-left: 40px;"> 1 AES-256 2 BLOWFISH 4 IDEA 8 Triple DES </div> Example (enable AES-256 and IDEA): <i>EnabledCiphers = 5</i>	15
AuditLogRetentionTime	Retention time in days for the records in audit log. All records older than specified will be deleted by housekeeping process.	90
BeaconHosts	Comma-separated list of hosts to be used as beacons for checking NetXMS server network connectivity. Either DNS names or IP addresses can be used.	Empty string
BeaconPollingInterval	Interval in milliseconds between beacon hosts polls.	1000
BeaconTimeout	Timeout in milliseconds to consider beacon host unreachable.	1000
CapabilityExpirationTime	Time before host's capability is considered to be expired. For example, if NetXMS agent on a host didn't respond within a specified time, the server will assume that there is no NetXMS agent on that host anymore.	604800
CheckTrustedNodes	Enable (1) or disable (0) checking of trusted nodes list for cross-node data collection (using <i>Proxy Node</i> DCI attribute).	1

Parameter	Description	Default
ClientListenerPort	TCP port used by the server to listen for incoming client connections.	4701
ConditionPollingInterval	Interval in seconds between polling (re-evaluating) of condition objects.	60
ConfigurationPollingInterval	Interval in seconds between configuration polls.	3600
DefaultCommunityString	System-wide default SNMP community string.	public
DefaultEncryptionPolicy	Set encryption policy for server to agent communications. Possible values are: 0 Encryption disabled 1 Encrypt connection only if agent requires encryption 2 Encrypt connection if agent supports encryption 3 Force encrypted connection	1
DeleteEmptySubnets	Enable (1) or disable (0) automatic deletion of subnet objects without any nodes within. When enabled, empty subnets will be deleted by housekeeping process.	0
DisableVacuum	Enable (0) or disable (1) automatic issue of VACUUM command by housekeeping process.	0
DiscoveryFilter	Name of NXSL script used as discovery filter. To disable filtering, set this parameter to <i>none</i> .	none
DiscoveryFilterFlags	Flags for system-generated discovery filter.	0
DiscoveryPollingInterval	Interval in seconds between network discovery polls.	900
EnableAdminInterface	Enable (1) or disable (0) local administrative interface (<i>nxadm</i> utility).	1
EnableAuditLog	Enable (1) or disable (0) audit log.	1
EnableEventStormDetection	Enable (1) or disable (0) detection of event storms.	0
EnableMultipleDBConnections	Enable (1) or disable (0) usage of multiple simultaneous connections to the database.	1
EnableSNMPTraps	Enable (1) or disable (0) receiving of SNMP traps.	1
EnableSyslogDaemon	Enable (1) or disable (0) receiving of syslog messages.	0
EnableZoning	Enable (1) or disable (0) support of management zones.	0
EventLogRetentionTime	Retention time in days for the records in event log. All records older than specified will be deleted by housekeeping process.	90
EventStormDuration	Number of seconds for which number of events per second should be higher then configured threshold to declare event storm condition.	15
EventStormEventsPerSecond	Minimal number of events per second needed to declare event storm condition.	100

Parameter	Description	Default
FixedStatusValue	Status value used by <i>Fixed</i> status propagation algorithm.	0
HouseKeepingInterval	Interval in seconds between housekeeper runs.	3600
IcmpPingSize	Size of ICMP packets (in bytes, excluding IP header size) used for status polls.	46
InternalCA	Enable (1) or disable (0) internal certificate authority.	0
KeepAliveInterval	Interval in seconds between sending keep alive packets to connected clients.	60
LockTimeout	Timeout in milliseconds for internal locks. It is not recommended to change this parameter, unless you are instructed by technical support to do so.	60000
LogAllSNMPTraps	Enable (1) or disable (0) logging of all incoming SNMP traps.	0
MailEncoding	Encoding for mails generated by NetXMS server.	iso-8859-1
NumberOfConditionPollers	Number of threads used for polling f condition objects.	10
NumberOfConfigurationPollers	Number of threads used for configuration polling. If you have many monitored nodes, you may need to increase the value of this parameter.	5
NumberOfDatabaseWriters	Number of threads used to perform delayed writes to database.	1
NumberOfDataCollectors	Number of threads used for data collection. If you have many DCIs configured, you may need to increase value of this parameter.	25
NumberOfDiscoveryPollers	Number of threads used for discovery polling. Normally you will not need to increase this parameter.	1
NumberOfRoutingTablePollers	Number of threads used for polling routing tables on monitored nodes. If you have really large number of monitored nodes (more than 1000), or if you have decreased routing table update interval, you may need to increase this parameter.	5
NumberOfStatusPollers	Number of threads used for status polling. Since accurate status polling is sensitive for normal system operation, it is highly recommended to have this parameter set to approximately 1/10 of number of monitored nodes.	10
NumberOfUpgradeThreads	Number of threads used to perform agent upgrades (i.e. maximum number of parallel upgrades).	10
PollCountForStatusChange	Number of consecutive unsuccessful polls required to declare node unreachable.	1
RADIUSNumRetries	Number of retries for RADIUS authentication.	5

Parameter	Description	Default
RADIUSPort	Port number used for connection to RADIUS server.	1645
RADISecret	Shared secret used for communication with RADIUS server.	netxms
RADIUSServer	Host name or IP address of RADIUS server.	localhost
RADIUSTimeout	RADIUS server response timeout in seconds.	3
ResolveNodeNames	Enable (1) or disable (0) automatic resolution of node IP addresses to DNS names during automatic network discovery process.	1
RetainCustomInterfaceNames	Retain (1) or not retain (0) custom interface names (don't overwrite them during configuration polls).	0
RoutingTableUpdateInterval	Interval in seconds between routing table polls.	300
RunNetworkDiscovery	Enable (1) or disable (0) automatic network discovery process.	0
SMSDriver	Mobile phone driver to be used for sending SMS.	<none>
SMSDrvConfig	SMS driver's parameters. For generic driver, it should be the name of COM port device.	Empty
SMTPFromAddr	Address used for sending mail from.	netxms@localhost
SMTPServer	SMTP server used for sending mail.	localhost
SNMPRequestTimeout	Timeout in milliseconds for SNMP requests sent by NetXMS server.	2000
StatusCalculationAlgorithm	Default status calculation algorithm. Possible values are: 1 Most critical 2 Single threshold 3 Multiple thresholds	1
StatusPollingInterval	Interval in seconds between status polls.	60
StatusPropagationAlgorithm	Default status propagation algorithm. Possible values are: 1 Unchanged 2 Fixed 3 Relative 4 Severity based	1
StatusShift	Status value shift for <i>Relative</i> status propagation algorithm.	0
StatusSingleThreshold	Threshold value multiplied by 100 for <i>Single Threshold</i> status calculation algorithm.	75
StatusThresholds	Threshold values for <i>Multiple Thresholds</i> status calculation algorithm. The value is a string of four two-digit hexadecimal numbers. Each number represents one of the thresholds, multiplied by 100. First number is <i>Warning</i> threshold, then <i>Minor</i> , <i>Major</i> , and <i>Critical</i> .	503C2814

Parameter	Description	Default
StatusTranslation	Translated status codes for <i>Severity based</i> status propagation. The value is a string of four two-digit hexadecimal numbers. Each number represents one translated status code. First number is for original <i>Warning</i> status, then for <i>Minor</i> , <i>Major</i> , and <i>Critical</i> .	01020304
SyncInterval	Interval in seconds between writing object changes to the database.	60
SyncNodeNamesWithDNS	Enable (1) or disable (0) synchronization of node names with DNS on each configuration poll.	0
SyslogListenPort	UDP port used by built-in syslog server.	514
SyslogRetentionTime	Retention time in days for records in syslog. All records older then specified will be deleted by housekeeping process.	90
ThresholdRepeatInterval	System-wide interval in seconds for resending threshold violation events. Value of 0 disables event resending.	0
TopologyDiscoveryRadius	Radius (maximum number of hops from seed device) for layer 2 topology discovery.	3
TopologyExpirationTime	Time to live in seconds for cached layer 2 topology information.	900
UseInterfaceAliases	Control usage of interface aliases (or descriptions). Possible values are: 0 Don't use aliases; 1 Use aliases instead of names when possible; 2 Concatenate alias and name to form interface object name.	0
WindowsConsoleUpgradeURL	URL pointing to actual version of NetXMS Console for Windows. Console application will try to download new version from this URL if it detects that upgrade is needed. You can use %version% macro inside the URL to insert actual server version.	http://www.netxms.org/download/netxms-%version%.exe

8.2 Agent Configuration

8.2.1 File: nxagentd.conf

File `nxagentd.conf` is a master configuration file for NetXMS agent. It contains all information necessary for agent's operation. Default location for this file is `/etc/nxagentd.conf` on UNIX systems and `C:\nxagentd.conf` on Windows.

The file can contain one or more parameters in `Parameter = Value` form, each parameter should be on its own line. Comments can be inserted after `"#"` sign. This file can also contain configuration for subagents. In this case, subagents' parameters should be placed in separate sections. Beginning of the section indicated by `"*"` sign, followed by a section name.

File can contain the following parameters (in main section):

Table 10: nxagentd.conf parameters

Parameter	Description	Default
Action	Define action, which can be later executed by management server. See the Agent Configuration section for detailed description of this parameter.	No defaults
ControlServers	List of management servers, which can execute actions on agent and change agent's config. Hosts listed in this parameter also have read access to the agent. Both IP addresses and DNS names can be used. Multiple servers can be specified in one line, separated by commas. If this parameter used more than once, servers listed in all occurrences will have access to agent.	Empty list
EnableActions	Enable (yes) or disable (no) action execution by agent.	yes
EnabledCiphers	Controls what ciphers agent can use for connection encryption. Value for this parameter is a cipher code. To enable more than one cipher, the codes should be summed up. Possible cipher codes: 1 AES-256 2 BLOWFISH 4 IDEA 8 Triple DES Example (enable AES-256 and IDEA): <i>EnabledCiphers = 5</i>	15
EnableProxy	Enable (yes) or disable (no) agent proxy functionality.	no
EnableSNMPProxy	Enable (yes) or disable (no) SNMP proxy functionality.	no
EnableSubagentAutoload	Enable (yes) or disable (no) automatic loading of platform subagent(s).	yes
ExternalParameter	Add parameter handled by external command. To add multiple parameters, you should use multiple ExternalParameter entries. See the Agent Configuration section for detailed description of this parameter.	No defaults

Parameter	Description	Default
FileStore	Directory to be used for storing files uploaded by management server(s).	/tmp on UNIX C:\ on Windows
ListenPort	TCP port to be used for incoming requests.	4700
LogFile	Agent's log file. To write log to syslog (or Event Log on Windows), use {syslog} as file name.	/var/log/nxagentd on UNIX C:\nxagentd.log on Windows
LogUnresolvedSymbols	If set to yes, all dynamically resolved symbols, which failed to be resolved, will be logged.	no
MasterServers	List of management servers, which have full access to agent. Hosts listed in this group can upload files to agent and initiate agent upgrade, as well as perform any task allowed for hosts listed in Servers and ControlServers. Both IP addresses and DNS names can be used. Multiple servers can be specified in one line, separated by commas. If this parameter used more than once, servers listed in all occurrences will have access to agent.	Empty list
MaxSessions	Maximum number of simultaneous communication sessions. Possible value can range from 2 to 1024.	32
PlatformSuffix	String to be added as suffix to the value of System.PlatformName parameter.	Empty string
RequireAuthentication	If set to yes, host connected to agent has to provide correct shared secret before issuing any command.	no
RequireEncryption	If set to yes, host connected to agent will be forced to use encryption, and if encryption is not supported by a remote host, the connection will be dropped. If agent was compiled without encryption support, this parameter has no effect.	no
Servers	List of management servers, which have read access to this agent. Both IP addresses and DNS names can be used. Multiple servers can be specified in one line, separated by commas. If this parameter is used more than once, servers listed in all occurrences will have access to agent.	Empty list
SessionIdleTimeout	Communication session idle timeout in seconds. If agent will not receive any command from peer within the specified timeout, session will be closed.	60
SharedSecret	Agent's shared secret used for remote peer authentication. If <i>RequireAuthentication</i> set to no, this parameter has no effect.	admin
StartupDelay	Number of seconds that agent should wait on startup before start servicing requests. This parameter can be used to prevent false reports about missing processes or failed services just after monitored system startup.	0

Parameter	Description	Default
SubAgent	Subagent to load. To load multiple subagents, you should use multiple SubAgent parameters. Subagents will be loaded in the same order as they appear in configuration file.	No defaults

Configuration file example:

```
#
# Sample agent's configuration file
#

MasterServers = 10.0.0.4
LogFile = {syslog}
SubAgent = winperf.nsm

# Below is a configuration for winperf subagent, in separate section

*WinPerf
EnableDefaultCounters = yes
```

8.3 Web Server (nxhttpd) Configuration

8.3.1 File nxhttpd.conf

File nxhttpd.conf is a master configuration file for NetXMS web server. It contains all information necessary for web server's operation. Default location for this file is `/etc/nxhttpd.conf` on UNIX systems and `C:\nxhttpd.conf` on Windows.

The file can contain one or more parameters in Parameter = Value form, each parameter should be on its own line. Comments can be inserted after "#" sign.

File can contain the following parameters:

Table 11: nxagentd.conf parameters

Parameter	Description	Default
DocumentRoot	Points to a directory which contains all static web content (images, scripts, etc.)	Windows: <code>var\www</code> under NetXMS installation directory UNIX: <code>share/netxms/www</code> under NetXMS installation directory
ListenPort	TCP port to be used for incoming requests.	<code>8080</code>
LogFile	Web server's log file. To write log to syslog (or Event Log on Windows), use <code>{syslog}</code> as file name.	<code>/var/log/nxhttpd</code> on UNIX <code>C:\nxhttpd.log</code> on Windows
MasterServer	NetXMS server to work with.	<code>localhost</code>
SessionTimeout	Web session timeout in seconds	<code>300</code>

8.4 Command Line Tools

8.4.1 Local Server Administration Tool (*nxadm*)

Administrator can use *nxadm* to access local NetXMS server console. This tool works only on the same machine where server is running. Server must have configuration parameter *EnableAdminInterface* set to 1 (which is the default setting).

Syntax:

```
nxadm -c command
nxadm -i
nxadm -h
```

If started with *-c* option, *nxadm* executes given command on NetXMS server console and exits immediately. With *-i* option *nxadm* enters interactive mode, and *-h* prints out help message.

8.4.2 Agent GET Tool (nxget)

This tool is intended to get values of parameters from NetXMS agent.

Syntax:

```
nxget [options] host [parameter [parameter ...]]
```

Where *host* is the name or IP address of the host running NetXMS agent, and *parameter* is a parameter or list name, depending on given options. By default, nxget will attempt to retrieve the value of one given parameter, unless given options override it.

Table 12: Valid options for nxget

Option	Description
-a <i>method</i>	Authentication method to be used. Valid methods are <i>none</i> , <i>plain</i> , <i>md5</i> , and <i>sha1</i> . Default is <i>none</i> .
-A <i>method</i>	Authentication method to be used for communications with proxy agent. Possible values are the same as for -a option.
-b	Get multiple parameters in one call — nxget will interpret all command line arguments after host name as separate parameter. Values will be printed out on separate lines, in the same order as parameters were specified.
-C	Get agent's configuration file instead of parameter(s) specified on command line. If this option is given, all command line arguments after host name will be ignored. Configuration file will be printed to standard output.
-e <i>policy</i>	Set encryption policy. Possible values are: <ul style="list-style-type: none"> 0 Encryption disabled 1 Encrypt connection only if agent requires encryption 2 Encrypt connection if agent supports encryption 3 Force encrypted connection Default value is 1. If requested encryption policy cannot be applied (for example, you request policy 3, but agent doesn't support encryption), connection to agent will fail.
-h	Display help and exit.
-i <i>seconds</i>	Get requested information continuously, with given interval. Communication session with the agent will not close between requests.
-I	Get list of parameters supported by agent instead of parameter(s) specified on command line. If this option is given, all command line arguments after host name will be ignored.
-K <i>file</i>	Server key file, required for establishing encrypted connections. Normally, nxget should know location of server key file without specifying it explicitly.
-l	Interpret parameter names given on command line as names of enums (parameters returning list of values).
-n	Show parameter name(s) in result. If this option is given, result will be printed in the following form: <pre>parameter = value</pre>
-O <i>port</i>	Proxy agent port number. Default is 4700.
-p <i>port</i>	Agent port number. Default is 4700.
-P <i>port</i>	Network service port (to be used with -s option).
-q	Quiet mode (print only results, no informational or error messages).
-r <i>string</i>	Service check request string.
-R <i>string</i>	Service check expected response string.
-s <i>secret</i>	Shared secret used for authentication.

Option	Description
-S <i>address</i>	Check state of network service at given address. If this option is given, all command line arguments after host name will be ignored.
-t <i>type</i>	Type of service to be checked. Possible values are: 0 Custom 1 SSH 2 POP3 3 SMTP 4 FTP 5 HTTP 6 Telnet Default value is 0.
-T <i>proto</i>	Protocol number for service check. Default is 6 (TCP).
-v	Display version and exit.
-w <i>seconds</i>	Command execution timeout. Default is 3 seconds.
-X <i>address</i>	Connect via proxy agent at given address.
-Z <i>secret</i>	Shared secret used for authentication to the proxy agent.

Some examples:

Get value of Agent.Version parameter from agent at host 10.0.0.2:

```
nxget 10.0.0.2 Agent.Version
```

Get value of Agent.Uptime and System.Uptime parameters in one request, with output in parameter = value form:

```
nxget -bn 10.0.0.2 Agent.Uptime System.Uptime
```

Get agent configuration file from agent at host 10.0.0.2:

```
nxget -C 10.0.0.2
```

Get value of parameter System.PlatformName from agent at host 10.0.0.2, connecting via proxy agent at 172.16.1.1:

```
nxget -X 172.16.1.1 10.0.0.2 System.PlatformName
```

Get value of enum Agent.SupportedParameters from agent at host 10.0.0.10, forcing use of encrypted connection:

```
nxget -e 3 -l 10.0.0.10 Agent.SupportedParameters
```

Check POP3 service at host 10.0.0.4 via agent at host 172.16.1.1:

```
nxget -S 10.0.0.4 -t 2 -r user:pass 172.16.1.1
```

8.4.3 NetXMS Database Manager (nxdbmgr)

This tool intended for NetXMS database maintenance.

Syntax:

```
nxdbmgr [options] check
nxdbmgr [options] init file
nxdbmgr [options] unlock
nxdbmgr [options] upgrade
```

Database Manager has three modes of operation: database checking, database initialization, and database upgrade. Mode of operation is selected by appropriate command line option.

Table 13: Command line options

Option	Description
check	Check the database for consistency. Use this mode to repair the database after server crash or incorrect shutdown.
init	Initialize empty database. You should provide name of the initialization file on the command line. Do not run this command if database already initialized!
unlock	Unlock database without further consistency checking. Normally, it is recommended to use <i>check</i> mode when you need to unlock the database. However, if you have to unlock database with format version not supported by current version of Database Manager , you should use this mode.
upgrade	Upgrade the database after NetXMS server upgrade.

Table 14: Valid options for nxdbmgr

Option	Description
-c <i>file</i>	NetXMS server configuration file to be used. Database Manager obtains the database connectivity parameters from the server configuration file.
-f	Force the database to unlock and repair without confirmation. Valid only in database checking mode.
-h	Display help and exit.
-v	Display version and exit.
-X	Ignore SQL errors when upgrading. It is not recommended to use this option, unless you are instructed to do so by NetXMS developers or support team.

8.5 NetXMS Scripting Language (NXSL)

8.5.1 Formal Grammar

```
script ::=
    module |
    expression

module ::=
    module_component { module_component }

module_component ::=
    function |
    use_statement

use_statement ::=
    use any_identifier ";"

any_identifier ::=
    IDENTIFIER |
    COMPOUND_IDENTIFIER

function ::=
    sub IDENTIFIER "(" [ identifier_list ] ")" block

identifier_list ::=
    IDENTIFIER { "," IDENTIFIER }

block ::=
    "{" { statement_or_block } "}"

statement_or_block ::=
    statement |
    block

statement ::=
    expression ";" |
    builtin_statement |
    ";"

builtin_statement ::=
    simple_statement ";" |
    if_statement |
    do_statement |
    while_statement |
    switch_statement |
    break ";"
    continue ";"

simple_statement ::=
    keyword [ expression ]

keyword ::=
    exit |
    print |
    return

if_statement ::=
    if "(" expression ")" statement_or_block [ else statement_or_block ]
```

```

while_statement ::=
    while "(" expression ")" statement_or_block

do_statement ::=
    do statement_or_block while "(" expression ")" ";"

switch_statement ::=
    switch "(" expression ")" "{" case { case } [ default ] "}"

case ::=
    case constant ":" { statement_or_block }

default ::=
    default ":" { statement_or_block }

expression ::=
    "(" expression ")" |
    IDENTIFIER "=" expression |
    expression ">" IDENTIFIER |
    "-" expression |
    "!" expression |
    "~" expression |
    inc IDENTIFIER |
    dec IDENTIFIER |
    IDENTIFIER inc |
    IDENTIFIER dec |
    expression "+" expression |
    expression "-" expression |
    expression "*" expression |
    expression "/" expression |
    expression "%" expression |
    expression like expression |
    expression ilike expression |
    expression "~=" expression |
    expression match expression |
    expression ismatch expression |
    expression "==" expression |
    expression "!=" expression |
    expression "<" expression |
    expression "<=" expression |
    expression ">" expression |
    expression ">=" expression |
    expression "&" expression |
    expression "|" expression |
    expression "^" expression |
    expression "&&" expression |
    expression "||" expression |
    expression "<<" expression |
    expression ">>" expression |
    expression "." expression |
    expression "?" expression ":" expression |
    operand

operand ::=
    function_call |
    type_cast |
    constant |
    IDENTIFIER

type_cast ::=
    builtin_type "(" expression ")"

builtin_type ::=
    int32 |

```

```
int64 |  
uint32 |  
uint64 |  
real |  
string
```

```
function_call ::=  
    IDENTIFIER "(" [ expression { "," expression } ] ")"
```

```
constant ::=  
    STRING |  
    INT32 |  
    INT64 |  
    UINT32 |  
    UINT64 |  
    REAL |  
    NULL
```

Terminal symbols:

```
IDENTIFIER ::= [A-Za-z_\$][A-Za-z_\$0-9]*  
COMPOUND_IDENTIFIER ::= { IDENTIFIER } (:: { IDENTIFIER }) +  
INTEGER ::= \-?(0x)?[0-9]+  
INT32 ::= INTEGER  
INT64 ::= { INTEGER } L  
UINT32 ::= { INTEGER } U  
UINT64 ::= { INTEGER } (UL|LU)  
REAL ::= \-?[0-9]+\.[0-9]+
```

8.5.2 Built-in Functions

8.5.2.1 abs

```
abs(number)
```

Returns the absolute value of *number*.

Examples:

```
abs(12.3)    ->    12.3  
abs(-0.307) ->    0.307
```

8.5.2.2 AddrInRange

```
AddrInRange(addr, start, end)
```

Checks if given IP address is within given range (including both bounding addresses). All IP addresses should be specified as strings. Function will return 0 if address is outside given range and 1 if inside.

Examples:

```
AddrInRange("10.0.0.16", "10.0.0.2", "10.0.0.44")    ->    1  
AddrInRange("10.0.0.16", "192.168.1.1", "192.168.1.100") ->    0
```

8.5.2.3 AddrInSubnet

```
AddrInSubnet(addr, subnet, mask)
```

Checks if given IP address is within given subnet (including subnet and broadcast addresses). All IP addresses should be specified as strings. Function will return 0 if address is outside given subnet and 1 if inside.

Examples:

```
AddrInSubnet("10.0.0.16", "10.0.0.0", "255.255.255.0")    ->    1
AddrInSubnet("10.0.0.16", "192.168.1.0", "255.255.255.0") ->    0
```

8.5.2.4 classof

```
classof(object)
```

Returns the class name for given object.

Examples:

```
classof($node)          ->    "NetXMS_Node"
```

8.5.2.5 gmtime

```
gmtime([time])
```

Converts time in UNIX format (number of seconds since epoch) to calendar date and time broken down into its components, expressed as UTC (or GMT timezone). Function uses either time given in *time* argument or current time if *time* is omitted. Return value is an object of class TIME with the following attributes:

Attribute	Synonim	Meaning
sec	tm_sec	Seconds after the minute
min	tm_min	Minutes after the hour
hour	tm_hour	Hours since midnight
mday	tm_mday	Day of the month
mon	tm_mon	Months since January
year	tm_year	Year
wday	tm_wday	Days since Sunday
yday	tm_yday	Days since January 1
isdst	tm_isdst	Daylight Saving Time flag

Examples:

```
gmtime(time())->year          ->    2008
```

8.5.2.6 left

```
left(string, length[, pad])
```

Returns a string of length *length*, containing the leftmost *length* characters of *string*. The string returned is padded with *pad* characters (or truncated) on the right as needed. The default pad

character is a blank. The *length* must be nonnegative.

Examples:

```
left("abc d",8)      ->  "abc d  "
left("abc d",8,".")  ->  "abc d..."
left("abc  def",7)   ->  "abc  de"
```

8.5.2.7 length

```
length(string)
```

Returns the length of a *string*.

Examples:

```
length("abcd")      ->    4
```

8.5.2.8 localtime

```
localtime([time])
```

Converts time in UNIX format (number of seconds since epoch) to calendar date and time broken down into its components. Function uses either time given in *time* argument or current time if *time* is omitted. Return value is an object of class TIME with the following attributes:

Attribute	Synonim	Meaning
sec	tm_sec	Seconds after the minute
min	tm_min	Minutes after the hour
hour	tm_hour	Hours since midnight
mday	tm_mday	Day of the month
mon	tm_mon	Months since January
year	tm_year	Year
wday	tm_wday	Days since Sunday
yday	tm_yday	Days since January 1
isdst	tm_isdst	Daylight Saving Time flag

Examples:

```
localtime(time())->year      ->    2008
```

8.5.2.9 lower

```
lower(string)
```

Translates given string to lowercase.

Examples:

```
lower("abCD")      ->    "abcd"
```

8.5.2.10 max

```
max(number[, number[, ...]])
```

Returns the largest number from the list specified.

Examples:

```
max(12, 6, 7, 9)  -> 12
max(-7, -3, -4.3) -> -3
```

8.5.2.11 min

```
min(number[, number[, ...]])
```

Returns the smallest number from the list specified.

Examples:

```
min(12, 6, 7, 9)  -> 6
min(-7, -3, -4.3) -> -7
```

8.5.2.12 pow

```
pow(x, y)
```

Calculates x raised to the power of y .

Examples:

```
pow(2, 3)  -> 8
```

8.5.2.13 right

```
right(string, length[, pad])
```

Returns a string of length *length*, containing the rightmost *length* characters of *string*. The string returned is padded with *pad* characters (or truncated) on the left as needed. The default pad character is a blank. The *length* must be nonnegative.

Examples:

```
right("abc d", 8)      -> "  abc d"
right("abc def", 5)     -> "c def"
right("17", 5, "0")     -> "00017"
```

8.5.2.14 SecondsToUptime

```
SecondsToUptime(seconds)
```

Converts given number of seconds to uptime string in format "*n* days, *hours:minutes*".

Examples:


```
SecondsToUptime(85)      -> "0 days, 00:01"
SecondsToUptime(3600)    -> "0 days, 01:00"
SecondsToUptime(93600)   -> "1 days, 02:00"
```

8.5.2.15 strftime

```
strftime(format, [time], [isLocal])
```

Formats a time string according to *format*. Function uses either time given in *time* argument (as a UNIX timestamp) or current time if *time* is omitted. Argument *isLocal* controls usage of time zone information – if it is set to TRUE (non-zero value) time will be converted to local time zone, otherwise UTC will be used.

The *format* argument consists of one or more codes; the formatting codes are preceded by a percent sign (%). Characters that do not begin with % are copied unchanged to output string. The formatting codes for **strftime** are listed below:

%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time representation appropriate for locale
%d	Day of month as decimal number (01 – 31)
%H	Hour in 24-hour format (00 – 23)
%I	Hour in 12-hour format (01 – 12)
%j	Day of year as decimal number (001 – 366)
%m	Month as decimal number (01 – 12)
%M	Minute as decimal number (00 – 59)
%p	Current locale's A.M./P.M. indicator for 12-hour clock
%S	Second as decimal number (00 – 59)
%U	Week of year as decimal number, with Sunday as first day of week (00 – 53)
%w	Weekday as decimal number (0 – 6; Sunday is 0)
%W	Week of year as decimal number, with Monday as first day of week (00 – 53)
%x	Date representation for current locale
%X	Time representation for current locale
%y	Year without century, as decimal number (00 – 99)
%Y	Year with century, as decimal number
%z, %Z	Time-zone name or abbreviation; no characters if time zone is unknown
%%	Percent sign

The # flag may prefix any formatting code. In that case, the meaning of the format code is changed as follows.

%#a, %#A, %#b, %#B, %#p, %#X, %#z, %#Z, %#%	# flag is ignored.
%#c	Long date and time representation, appropriate for current locale. For example: "Tuesday, March 14, 1995, 12:41:29".
%#x	Long date representation, appropriate to current locale. For example: "Tuesday, March 14, 1995".
%#d, %#H, %#I, %#j, %#m, %#M, %#S, %#U, %#w, %#W, %#y, %#Y	Remove leading zeros (if any).

Examples:

```
strftime("%H:%M")      -> "07:24"
strftime("%#H:%M")     -> "7:24"
strftime("%d-%b-%Y %H:%M:%S %Z", 1178007761) ->
                                "01-May-2007 11:22:41 FLE Daylight Time"
strftime("%d-%b-%Y %H:%M:%S", 1178007761, 0) -> "01-May-2007 08:22:41"
```

8.5.2.16 substr

```
substr(string, n[, len])
```

Returns the substring of string that begins at the *n*th character and is of length *len*. The *n* must be a positive whole number. If *n* is greater than length(*string*), then empty string is returned. If you omit length, the rest of the string is returned.

Examples:

```
substr("abcdef", 2, 2)      -> "cd"
substr("abcdef", 8)        -> ""
substr("abcdef", 3)        -> "def"
```

8.5.2.17 time

```
time()
```

Gets the system time. Function returns the number of seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time, according to the system clock.

8.5.2.18 typeof

```
typeof(value)
```

Returns the data type for given value. Type is returned as lowercase string. The following type names can be returned:

- null
- object
- string
- real
- int32
- int64
- uint32
- uint64

Examples:

```
typeof("abc")      -> "string"
typeof(17)         -> "int32"
typeof(17000000000) -> "int64"
```

8.5.2.19 upper

```
upper(string)
```

Translates given string to uppercase.

Examples:

```
upper("abCD")    ->    "ABCD"
```

8.5.3 Functions Available in DCI Transformation Scripts

8.5.3.1 FindDCIByDescription

```
FindDCIByDescription(node, description)
```

Find DCI by description (search is case-insensitive). Function returns DCI ID on success or 0 if DCI with matching description was not found. Parameter *node* is a node object, you can use predefined variable *\$node* to refer to current node.

Examples:

```
FindDCIByDescription($node, "Status")    -> 4
FindDCIByDescription($node, "bad dci")    -> 0
```

8.5.3.2 FindDCIByName

```
FindDCIByName(node, name)
```

Find DCI by name (search is case-insensitive). Function returns DCI ID on success or 0 if DCI with matching description was not found. Parameter *node* is a node object, you can use predefined variable *\$node* to refer to current node.

Examples:

```
FindDCIByName($node, "Agent.Uptime")    -> 5
FindDCIByName($node, "bad")              -> 0
```

8.5.3.3 GetDCIValue

```
GetDCIValue(node, id)
```

Get last value of DCI with given id on given node. Function returns last DCI value on success or NULL in case of error (for example, DCI with given id does not exist or has no collected values). Parameter *node* is a node object, you can use predefined variable *\$node* to refer to current node.

Examples:

```
GetDCIValue($node, FindDCIByName($node, "Status"))    -> 0
GetDCIValue($node, FindDCIByName($node, "Agent.Version")) -> "0.2.20"
GetDCIValue($node, 12345)                              -> NULL
```

8.5.4 Functions Available in Event Processing Scripts

8.5.4.1 FindSituation

```
FindSituation(situation_name_or_id, instance)
```

Find situation instance either by situation object name and instance name or by situation object id and instance name (name search is case-insensitive). Function returns situation instance object on success or NULL if referred situation instance was not found.

Examples:

```
FindSituation("my_situation", "my_instance")      -> valid_object
FindSituation(1, "my_instance")                  -> valid_object
FindSituation("bad_situation_name", "my_instance") -> NULL
```

8.5.4.2 GetSituationAttribute

```
GetSituationAttribute(situation, attribute)
```

Get current value of situation's instance's *situation* attribute with name *attribute*. Function returns current attribute value on success or NULL in case of error (for example, attribute with given name does not exist or invalid object given). Parameter *situation* is a situation instance object, usually returned by FindSituation function.

Examples:

```
GetSituationAttribute(s, "status")      -> "current value"
GetSituationAttribute(s, "bad attribute") -> NULL
```

Notes:

If attribute name conforms to NXSL identifier naming conventions, you can access it directly as situation instance object attribute. For example, the following call to GetSituationAttribute:

```
GetSituationAttribute(sobj, "status")
```

is equal to:

```
sobj->status
```

8.6 Macros for Event Processing

On various stages of event processing you may need to use macros to include information like event source, severity, or parameter in your event texts, alarms, or actions. You may use the following macros to accomplish this:

%n	Name of event source object.
%a	IP address of event source object.
%i	Unique ID of event source object.
%t	Event's timestamp is a form <i>day-month-year hour:minute:second</i> .
%T	Event's timestamp as a number of seconds since epoch (as returned by time() function).
%c	Event's code.
%s	Event's severity code as number. Possible values are: 0 Normal 1 Warning 2 Minor 3 Major 4 Critical
%S	Event's severity code as text.
%v	NetXMS server's version.
%u	User tag associated with the event.
%m	Event's message text (meaningless in event template).
%A	Alarm's text (can be used only in actions to put text of alarm from the same event processing policy rule).
%1 - %99	Event's parameter number 1 .. 99.
%%	Insert % character.

If you need to insert special characters (like carriage return) you can use the following notations:

\t	Tab character
\n	CR/LF character pair
\\	Backslash character