Nexthink V6.19

API and Integrations

Generated: 12/11/2018 9:23 pm

Table of Contents

Integrating with Nexthink

Overview

Nexthink collects and processes a great deal of information coming from your IT infrastructure. Nexthink is able to monitor, including but not limited to, the configurations, program installations, application executions, user interactions, network connections, printer usage and system failures of the machines inside your corporate network. In this way, Nexthink provides you an end-user perspective of what is going on inside your IT infrastructure. This data is highly valuable to any IT department. However, IT departments typically do not use just one tool, but multiple tools for different purposes. The ability to combine the available tools in a convenient way is a key factor to the efficiency of your IT department.

Hereby we explain the built-in mechanisms of Nexthink to interact and share data with third-party tools by means of standard protocols and common interchange formats. Your IT team will then be enabled to build full-blown IT solutions by taking advantage of the monitoring capabilities of Nexthink and integrating them into the third-party software of your choice.

Getting data through the Web API

Overview of the Web API

The Web API is the main interface that Nexthink offers for integrating Nexthink data with external information systems. While Finder investigations provide a user with the means to query the Nexthink database, investigations are not well adapted to be launched and processed by external programs. The Web API fills this gap by offering third-party applications a standard programming interface to query the Nexthink database. The Engine implements the Web API as a RESTful web service over HTTPS. As such, the Web API can accept requests from any external application that supports the HTTP protocol over TLS/SSL (HTTPS). The default port for connecting to the Web API is 1671. Since the Web API uses well-established Internet protocols for communicating with external applications, many tools support them directly. Information systems such as Configuration Management Databases or Issue Tracking Systems are typically able to access RESTful web services. These systems can quickly benefit from the integration of Nexthink data by querying the Web API.

Starting from Nexthink V5.3, the Web API comes in two flavors:

Web API V2
: The newest and most flexible version of the Web API. Build advanced queries using NXQL, the Nexthink Query Language, to satisfy your most demanding integration needs. Send queries using either the GET or POST methods of the HTTPS protocol and receive the results of your queries in the format of your choice: XML, JSON, HTML or CSV.

Web API V1 (deprecated)
: The simple way to turn your Finder investigations into web-accessible queries. Build and publish your queries visually with the tools provided by the Finder. Access to those queries using the GET method of the HTTPS protocol and get results in XML format.

Prerequisites

In order for the protocols of the Web API to work, set the External DNS name of the Engine to an appropriate value.

If your Engine is behind a Firewall, remember to open access to the default TCP port for the Web API (1671), or to the port number that you have configured instead.

Related tasks

- Introducing the Web API V2
- NXQL Tutorial
- Publishing an investigation (Web API V1)
- Setting the names of the Engines

Bidirectional integration with the Finder

Overview

The Finder is a user-friendly graphical interface to the Nexthink database. As such, the integration with the Finder is not based on sharing data with external applications (the Web API already covers that part), but on interacting with other applications. The Finder can be launched from external tools in an automated way and it is capable of triggering specific actions on external applications as well. The Finder interacts with other applications by means of the nxt application protocol and custom actions.

The nxt application protocol

The *nxt application protocol* provides you with the means to launch the Finder and perform some specific actions on it by just stating a URL. The Finder registers the nxt protocol in Windows during its installation. From that point on, Windows recognizes the URI scheme **nxt**, associating it to the Finder application. You can embed **nxt** URLs as hyperlinks in HTML web pages, use them directly in the address bar of your web browser, or launch them from the Run dialog box of Windows.

There are seven types of actions that the Finder can handle when called from an **nxt** URL:

- Open a new Finder.
- Display the device view.
- Display the user view.
- Display the service view.
- Edit a metric.
- Edit a category.
- Launch an arbitrary investigation.

The nxt protocol offers a mechanism to specify both the Portal and the Engine to which the Finder must connect, as well as the name of the Finder user for the connection.

Open a new Finder

The simplest action that can be triggered with the nxt protocol is to open a new instance of the Finder:

```
nxt://New-NxFinder
```

Display the Device View

This command of the nxt protocol opens the device view of a particular device. Identify the device either by its name or its last known IP address. Starting from V6.18, the use of the ID to identify the device is deprecated, although still functional.

```
nxt://Show-NxSource?Name=DEVICE_NAME

nxt://Show-NxSource?IpAddress=SOURCE_LAST_IP_ADDRESS

nxt://Show-NxSource?Id=DEVICE_ID
```

By default, the Device View displays the last 24 hours of the device. Optionally, specify a different range of dates for the Device View with the parameters **StartDate** and **EndDate**:

```
nxt://Show-NxSource?Name=SOURCE_NAME&StartDate=START_DATE&EndDate=END_DATE
```

The dates must be expressed in the UTC time zone with the format: YYYY-MM-JJ**T**hh:mm. For example: 2016-04-04T12:00. The time span between the **StartDate** and the **EndDate** must be strictly smaller than 7 days.

Display the User View

Use this command to open the user view of a particular user in the Finder. Identify users by their name:

```
nxt://Show-NxUser?Name=USER_NAME
```

By default, the User View displays the last 24 hours of the user. Optionally, specify a different range of dates in the same way as explained for the Device View above.

Display the Service View

The following command of the nxt protocol lets you open the service view for a given service in the Finder:

```
nxt://Show-NxService?name=SERVICE_NAME
```

Replace SERVICE_NAME by the actual name of the service that you want to monitor, paying attention to capital letters because this argument is case sensitive.

Edit a metric

To open the Finder for editing a particular metric, build a nxt protocol URL with the following command and provide the name of the metric as parameter:

```
nxt://Edit-NxMetric?Name=METRIC_NAME
```

Note that the names of metrics are case sensitive.

Edit a category

To open the Finder for editing a particular category, build a nxt protocol URL with the following command:

```
nxt://Edit-NxCategory?Name=CATEGORY_NAME&Type=CATEGORY_TYPE
```

Replace CATEGORY_NAME by the name of the category that you want to edit and CATEGORY_TYPE by the type of object to which the category applies: application, binary, destination, device, domain, executable, package, port, printer, or user.

Launch an investigation

Using the nxt protocol, you may also run an arbitrary investigation in the Finder. The command that you need to use for launching an investigation is the following:

```
nxt://Run-NxInvestigation?Encoding=ENCODING_FORMAT&InvestigationXml=INVESTIGATION_XML
```

The investigation is specified in XML format. You can get the XML representation of an investigation from the Finder by right-clicking the name of the investigation and selecting the option **Export**. You may then choose to export the investigation to the clipboard or to a file. In any case, you get the investigation in its XML form.

Note that the XML of an investigation contains special characters that are not supported by URLs. Solve by properly encoding the investigation by setting the parameter Encoding to **Url** or **Base64** (see the section Encoding the arguments of an nxt URL). Find below the same investigation encoded in the two formats. Note that parameters are encoded.

Example of **Url** encoding:

```
nxt://Run-NxInvestigation?Encoding=Url&Host=192.168.5.5&Port=443&
InvestigationXml=%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22utf-16
%22%3F%3E%3CInvestigation%20xmlns%3Axsi%3D%22http%3A%2F%2Fwww.w3.org
%2F2001%2FXMLSchema-Instance%22%20xmlns%3Axsd%3D%22http%3A%2F%2Fwww.w3.org
%2F2001%2FXMLSchema%22%20DataModelVersion%3D%228%22%20SyntaxVersion%3D%22
2%22%3E%3CLabel%3Etest%3C%2FLabel%3E%3CObject%3Esource%3C%2FObject%3E%3C
Description%20%2F%3E%3CFieldList%3E%3Cstring%3Ename%3C%2Fstring%3E%3C%2F
FieldList%3E%3CCategoryList%20%2F%3E%3CAggregateList%20%2F%3E%3C
ObjectConditionList%20%2F%3E%3C%2FInvestigation%3E
```

Example of **Base64** encoding:

```
nxt://Run-NxInvestigation?Encoding=Base64&Host=MTkyLjE2OC41LjU=&Port=NDQz&
InvestigationXml=PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTE2Ij8+PEludmVzdGl
nYXRpb24geG1sbnM6eHNpPSJodHRwOi8vd3d3LnczLm9yZy8yMDAxL1hNTFNjaGVtYS1JbnN0YW5jZSI
geG1sbnM6eHNkPSJodHRwOi8vd3d3LnczLm9yZy8yMDAxL1hNTFNjaGVtYSIgRGF0YU1vZGVsVmVyc2l
vbj0iOCIgU3ludGF4VmVyc2lvbj0iMiI+PExhYmVsPnRlc3Q8L0xhYmVsPjxPYmplY3Q+c291cmNlPC9
PYmplY3Q+PERlc2NyaXB0aW9uIC8+PEZpZWxkTGlzdD48c3RyaW5nPm5hbWU8L3N0cmluZz48L0ZpZWx
kTGlzdD48Q2F0ZWdvcnlMaXN0IC8+PEFnZ2F0ZUxpc3QgLz48T2JqZWN0Q29uZGl0aW9uTGlzdCA
vPjwvSW52ZXN0aWdhdGlvbj4=
```

Note that, for the links to fit the page width, the examples above include line breaks. To test them, remove the line breaks when copying the URLs or copy the links from the following web page:

- NXT protocol test

Establishing the connection

If you do not provide connection details to the nxt protocol, the Finder either executes the action in the context of the current session (if a running Finder is available with a session already established), or asks the user to open a new session (by displaying the login dialog) and then executes the action.

Alternatively, state the connection details as parameters in the URI:

Host
: The DNS name or IP address of the Portal.

Port
: The port number where the Portal listens at Finder connections (443 by default).

UserName (optional)
: The name of the Finder user to impersonate for the connection.

EngineName (optional)
: The name of the Engine to select.

The Finder opens the first session that matches the connection details. If you do not provide an Engine name, the Finder displays the Engine selection dialog (unless there is only one Engine or the user has a favorite Engine). If you do not provide the user name, the Finder opens the first matching session regardless of whom the user is.

For instance, to open the device view on a particular connection:

```
nxt://Show-NxSource?Name=SOURCE_NAME&Host=PORTAL_ADDRESS&Port=PORT_NUMBER&UserName=USER
```

For backwards compatibility with V5, you can supply a session name to the nxt protocol in place of the connection details. Note however that, in V6, a session defines a connection between the Finder and a Portal; whereas in V5, a session defines a connection between the Finder and an Engine. Therefore, in a multi-Engine V6 setup, specifying the session name may not be enough to completely describe the connection: the Finder knows about the targeted Portal, but not about the Engine. In that case, the Finder usually displays the Engine selection dialog. Only if the user has a favorite Engine for the session (or in single Engine setups), the Finder skips the Engine selection step. Thus, the parameter **SessionName** is deprecated in V6.

To open a device view from a particular session, write the following URI:

```
nxt://Show-NxSource?Name=SOURCE_NAME&SessionName=SESSION_NAME
```

To prevent the Finder from asking for user credentials, use those sessions or connection details for which you have saved the password. Alternatively, if you have enabled Windows authentication in your setup, you can instruct the nxt protocol to use it by setting the parameter **UseSso** to *true*:

```
nxt://Show-NxSource?Name=SOURCE_NAME&Host=PORTAL_ADDRESS&Port=PORT_NUMBER&UseSso=true
```

When using Windows authentication, keep in mind that the Portal address must be a proper DNS name and not an IP address.

Creating nxt protocol links from the Finder

Nxt protocol links are very useful, for instance, in dashboard descriptions to offer the possibility of configuring a dashboard (edit related metrics or categories), or simply to complete the dashboard with complementary information displayed in the Finder. Writing a link for the nxt protocol, however, may be a cumbersome task, specially when you need to encode an investigation. To make this task easier for you, it is possible to create nxt protocol links for some actions directly from the Finder.

Generate nxt protocol links from the Finder for the following actions:

- Launch an investigation
- Edit a category
- Edit a metric
- Display the service view

To easily create nxt protocol links from the Finder:

1. Right-click the name of an investigation, category, metric, or investigation in the left-hand side accordion menu.
2. Select **Export** from the context menu. Depending on the kind of item that you right-clicked, select:
    - **Run investigation URL to clipboard**, if you chose an investigation. When the resulting URL is longer than 2083 characters, the Finder displays a message to warn you that some browsers might not support this kind of link (see the limitations of the nxt protocol).
    - **Edit category URL to clipboard**, if you chose a category.
    - **Edit metric URL to clipboard**, if you chose a metric.
    - **View service URL to clipboard**, if you chose a service.
3. Paste the URL from the clipboard and share it in a web page, email, or dashboard description.

Limitations of the nxt protocol

Investigations in XML form can be quite verbose. The more conditions you add to an investigation, the longer the XML becomes. However, the maximum supported length for an nxt URL is limited to 2083 characters. Therefore, you may not be able to use this method to launch complex investigations.

Note that the limit in the number of characters of a URL can be even more restrictive depending on the browser that you use to launch the request. For instance, Internet Explorer supports a maximum of 507 characters.

Encoding the arguments of an nxt URL

In the case that the arguments of an nxt URL contain special characters which are not supported by URLs, you may encode them using Base64 or URL (percent) encoding. In order to specify the encoding method, you must include an additional Encoding argument as the first argument of the nxt URL. This argument can take either one of two values: Base64 or Url. Please note that once you have chosen an encoding method, all the arguments of the URL must be encoded using that method. It is not possible to mix different encoding methods in the same nxt URL.

Base64 encoding

Whenever possible, it is recommended to use Base64 encoding for nxt URLs, as it is more robust. This method prevents double enconding or double decoding scenarios that may appear with URL encoding. The disadvantage of this method is that arguments become unreadable to humans. For example, the folowing

URL instructs the Finder to display a device with id 12:

```
nxt://Show-NxSource?Encoding=Base64&Id=MTI=
```

URL encoding

URL encoding is a simple alternative to Base64 encoding that ensures support for limited scenarios. URL encoding can be used for instance when one of the arguments contains a space character. Some browsers in fact automatically encode a space in a URL as "%20". The following hyperlink:

```
<a href="nxt://Show-NxSource?Name=Work PC1">My link</a>
```

when invoked from such browsers is translated into:

```
nxt://Show-NxSource?Name=Work%20PC1
```

with the consequence that, if no encoding is specified, the system will look for a device with name *Work%20PC1* instead of *Work PC1*. The following example shows how to correct such an issue using URL encoding:

```
<a href="nxt://Show-NxSource?Encoding=Url&Name=Work%20PC1">My link</a>
```

Information levels

Finder sessions are bound to Finder user accounts. Depending on the information level of the user account that is bound to a given session, you may or may not be able to perform a particular query to the Engine using the nxt protocol. As a guideline, the following table shows the variants of the Show-NxSource command which are available depending on the information level of the Finder account that the session provided is using to connect to the Engine.

Information Level	Show Source by Name	Show Source by IP Address	Show Source by ID
Anonymous Users and Devices	Not Available	Not Available	Available
Anonymous Users	Not Available	Available	Available
none (full access)	Available	Available	Available

Testing and debugging nxt protocol invocations

When invoking a malformed nxt URL with a wrong command, argument or encoding, the nxt protocol handler terminates silently without displaying any error message. During integration, however, it is useful to have some feedback and

know why an invocation failed. A possibility is to attach a trace listener to the protocol handler.

Create a file named **Nexthink.Finder.PowerShell.exe.config** with the content below and save it to the folder where the **Nexthink.Finder.Powershell.exe** file is found (the **Integration** directory under the installation directory of the Finder):

```
<?xml version="1.0"?>
<configuration>
 <system.diagnostics>
 <trace autoflush="true" indentsize="4">
 <listeners>
 <add name="FileListener"
 type="System.Diagnostics.TextWriterTraceListener"
 initializeData="DESTINATION_FILE" />
 </listeners>
 </trace>
 </system.diagnostics>
</configuration>
```

where DESTINATION_FILE is the full path of the log file where trace information will be saved (for instance, c:\log\Finder_Launcher.log).

Custom actions

Custom actions let the user launch external operations from the Finder. In that sense, custom actions are complementary to the nxt application protocol, which lets the Finder be automated.

Custom actions are applied within the context of an object, an activity, or an event. Note that, when defining custom actions, any of these items is named the *object* of the action. Therefore, the object of a custom action can be not only a device, a user, a printer... but also a connection, an execution, or a device warning. A custom action object is thus anything on which we can set an investigation. In addition to specifying an object, a custom action requires the user to specify an attribute or a category of the object. The value of the attribute may later be used as an argument to the custom action.

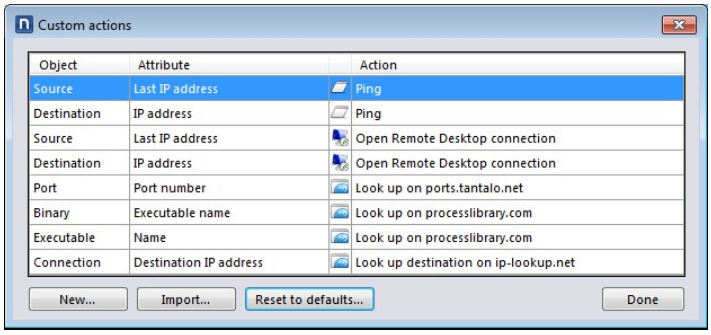
There are three types of custom actions available:

1. Open a URL
2. Run a command in the Command Prompt
3. Run an external program

The Finder stores custom actions locally in the machine where the Finder was installed. Therefore, your set of defined custom actions will always be available independently of the Engine that you are connecting to. You may also export your set of custom actions in order to share them among different Finder installations.

Default custom actions

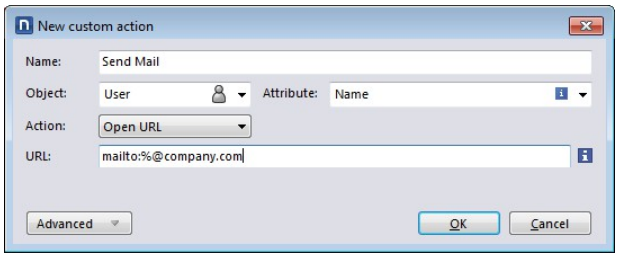
Nexthink Finder comes with a default set of useful custom actions. With the default custom actions, you can ping a machine, open remote desktop connections, or look up for information about processes, ports and IPs in well-known web sites. Set of default custom actions.



User-defined custom actions

You may extend the set of contextual actions available by defining your own custom actions. As an example, we are going to create a custom action for the user object, so we can automatically send a mail to a specific user. We start by opening the set of available custom actions by clicking on the Tools option in the menu and then selecting Custom actions....

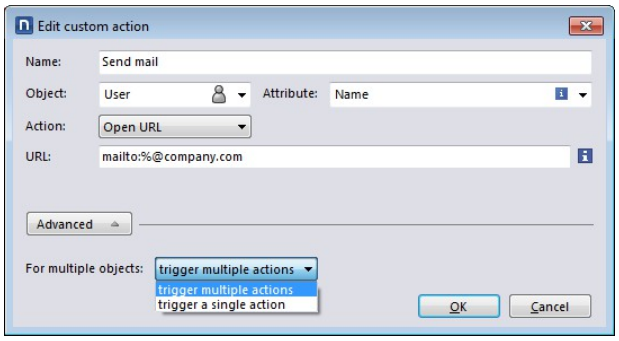
If this is the first custom action that you create, you will see the same set of default actions that we saw in the previous chapter. We just clik on New? and a dialog for creating our new custom action will appear. We fill in the dialog with the following values:



The percent character "%" is replaced at the execution of the custom action by the attribute that we selected. In this case, the name of the user will replace the % character. If you need to write a % character in the command that you do not want to be replaced, use a double percent: %%. Please note that this is a simplified example and that we are assuming that we can directly assemble the email address of a user just by concatenating the name of the user and the name of the company. We have used the Open URL action together with the mailto scheme in order for the system to launch your default email composer when the action is executed.

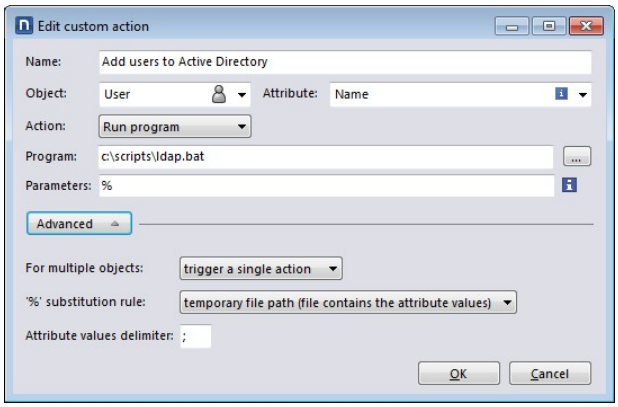
Custom actions can be applied to one or several objects at the same time. When editing a custom action, we can decide if we want the action to be applied separately to each one of the objects selected or if we want to execute the action over all of the selected objects at once.

This option can be set by clicking on the Advanced section of the edit dialog of a custom action. In our case, since we have selected the default trigger multiple actions, when multiple users are selected an email will be sent separately to each one of the users. If trigger a single action is selected, the ?%? character will be substituted for the concatenated attribute values of all the objects selected and the action will be executed only once. You may specify as well a value delimiter to separate each one of the attribute values. By default, the delimiter character is the semicolon ?;?.



When triggering a single action for multiple objects, the concatenation of many attribute values may yield a very long chain of characters to substitute the ?%? sign. If your action consists on running a command based on a very long parameter, you may run out of space in the command line. In order to overcome this limitation, there is an additional option in the Finder (starting from version 4.3.3) which lets you save the concatenated parameter in a temporary file.

Thus, only the path of the temporary file replaces the placeholder ?%?, as in the following example.



Executing custom actions

You can invoke custom actions from the context menu of an object or a set of objects. You can select the objects either from the List result of an investigation or from the Network activity or Local activity views. Note that the Network activity and Local activity views may or may not be available depending on the specific kind of object.

Exporting data from the Finder

The Finder also includes a way to share data with external applications through the clipboard. The results of an investigation may be partially or entirely copied to the clipboard. You just have to right click on the selected objects and choose the option Copy rows. Then you may paste the contents of the clipboard into your favourite spreadsheet application.

Instead of copying the whole rows of your selection, you may just copy to the clipboard the value of the attribute which is below the mouse cursor when you do the right-click. In the example above, the context menu shows that you can copy the name of the first computer. Since this method requires user intervention, it is not adapted to be automated. As we said above, if you regularly need to query the Nexthink database from an external application, the Web API is the recommended methodology.

Triggering remote actions via their API

Overview

The API of remote actions makes possible to trigger remote actions programmatically, enabling their integration with third-party products such as

self-service portals or ticketing systems.

The API of remote actions is exposed by the Portal as a REST API.

Applies to platforms:

Prerequisites

For a remote action to be triggered through the API, the following prerequisites apply:

- The remote action must allow manual triggering.
- The remote action must be triggered on behalf of a user whose profile includes Finder access with the option **Allow API of remote actions** ticked.

Calling the API

The Portal exposes the API of remote actions as a REST API under the URL:

```
https://[portal.company.com]/api/remoteaction/v1/run
```

In the URL, substitute `[portal.company.com]` for the external DNS name of your Portal.

To trigger a remote action, submit a POST request to the URL of the API (note that GET requests are not supported, returning a 404 error) with a JSON payload containing two parameters:

Name	Description
`RemoteActionUid`	Identifier of the remote action
`DeviceUids`	List of device identifiers

Example of the JSON payload of a request to the API of remote actions:

```
{
  "RemoteActionUid": "b21b9377-3624-4046-8378-76244657d2d4",
  "DeviceUids" :
["da581fd269e0bc4c03b767a345fbf5d3","fa623653cd663e19a58d69519471bbef"]
}
```

The call is dispatched to all the Engines connected to the Portal and executed asynchronously, meaning that it returns immediately after the request has been

validated. A successful response from the Portal does not guarantee the execution of the remote action on the selected devices. For unsuccessful responses, see the list of error conditions below.

Obtaining the UIDs of remote actions and devices

To get the UID of a remote action:

1. Log in to the Finder as a user with the permission to edit remote actions.
2. Locate the desired remote action in the **Remote actions** section of the left-hand side panel.
3. Right-click the remote action name.
4. Select **Export > Remote action to clipboard**.
5. Open your favorite XML or plain text editor.
6. Press **Ctrl+V** to paste the contents of the clipboard on the editor.
7. Find the **UID** attribute of the **Action** element in the XML of the remote action.

Find the UIDs of the devices through either:

- The Finder: Display field **UID** of the device object.
- NXQL: Retrieve the **device_uid** field of the device objects, for instance:

```
(select device_uid (from device))
```

HTTP Headers

Send your POST request to the API of remote actions with the following HTTP headers to specify JSON content and basic authentication:

```
Content-type: application/json
Authorization: Basic [base-64(user:password)]
```

Replace `[base-64(user:password)]` with the credentials (in base-64 encoding) of a Nexthink user that has the right to access the API of remote actions.

Error conditions

In response to a request, the Portal may send one of the following answers when something is wrong with the request:

Error type	HTTP code	Cause
	Unauthorized 401	

Access Denied	Forbidden 403	• Authentication error • User with insufficient permissions to run the specified remote action
Validation error	Bad request 400	• Invalid JSON • Invalid encoding • Invalid Content-type • Invalid or missing UID of remote action • Invalid or missing UIDs of devices • Unknown or disabled remote action • Manual execution of remote action not allowed • Too many device UIDs specified (limit of 12 000)
Unknown error	Internal server error 500	• Undefined internal error

Related tasks

- Triggering a remote action manually

Integrating investigation-based alerts

Overview

In this section, learn about the notifications generated by investigation-based alerts to integrate them with other systems.

Investigation-based alerts return a set of objects matching the specified conditions either immediately or periodically, sending the result via email or, in the case of global alerts, optionally via the system log.

Create and configure an investigation-based alert using the Finder. The account used to create the alert has an influence on the mechanisms to notify it. If the account is properly configured with a valid email address, alerts associated with that account will send emails to the configured address. In addition to the configured email address, you may specify other recipients of the alert email in the dedicated space. If no email address has been configured for that particular account, at least one recipient has to be manually specified in the dedicated space.

Only those users with the appropriate profile setting (**Allow system configuration**) can create global alerts. Global alerts can be sent via email, as described above, and optionally via the system log.

Email integration of investigation-based alerts

Email is a proven, ubiquitous and mature technology, and thus a suitable means to integrate alert info into third-party software. Email is also easy to automate, since many programming languages have libraries available to send and receive email by means of standard email protocols such as SMTP, IMAP or POP.

Investigation-based alerts are sent via email in HTML form, using the UTF-8 charset and base64 transfer encoding. The subject of the message consists of the word Nexthink followed by a colon and then the name of the alert. The message content is composed of two HTML tables preceded by an embedded CSS snippet which defines the style of the two tables. The first HTML table displays some general information about the alert, whereas the second HTML table holds the result of the investigation associated to the alert, in the case of investigation-based alerts. If an investigation-based alert fails to execute, a message indicating the reason for the failure appears in the place of the results of the corresponding investigation.

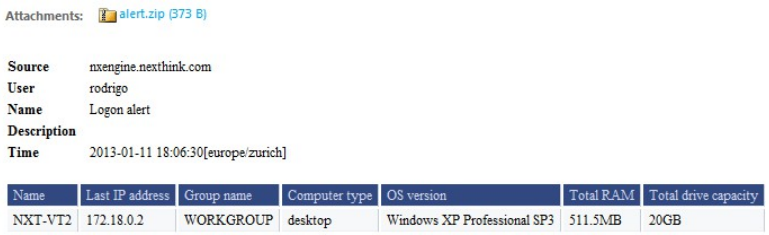
In addition to the HTML table with the results of the investigation, the email of an investigation-based alert includes an attachment particularly well suited for integration. This is a compressed Comma Separated Values (CSV) file that holds the same results shown by the HTML, but in plain text. CSV files are understood by a great number of different tools and they are very easy to parse programmatically.

HTML info table

The HTML info table is composed of five fields which give general information about the context of the alert:

- **Source**: name of the Engine that generated the alert.
- **User**: name of the Finder account associated to the alert.
- **Name**: the name of the alert itself.
- **Description**: brief description of the alert, as displayed in the Finder.
- **Time or Period**: For non-periodic (system or immediate) alerts, the time at which the alert was triggered is shown. In the case of periodic alerts, the period for which the alert was computed is displayed. In both cases, the time of the day or interval of time is expressed in the timezone of the associated user. The name of the timezone is displayed right after the

corresponding time or period.



Attachments: alert.zip (373 B)

Source	nxengine.nexthink.com
User	rodrigo
Name	Logon alert
Description	
Time	2013-01-11 18:06:30[europe/zurich]

Name	Last IP address	Group name	Computer type	OS version	Total RAM	Total drive capacity
NXT-VT2	172.18.0.2	WORKGROUP	desktop	Windows XP Professional SP3	511.5MB	20GB

HTML results table

The results of an investigation-based alert are displayed in the form of a HTML table whose first row holds the names of the fields that were selected during the configuration of the alert. Up to a maximum of fifteen fields will be displayed in an email of an alert. If more than fifteen fields were selected when editing the investigation associated to the alert, only the first fifteen will appear in the email and the rest will be discarded. The CSS included in the mail makes the first row of the HTML results table to be highlighted, so the names of the selected fields appear as the headers of each column. Each subsequent row holds the values of the fields for every alerted object, that is, each row shows information about an object which met the conditions specified by the alert. The maximum number of alerted objects which can be displayed in the email of of an investigation-based alert is 250 objects. Therefore, a HTML results table may have a maximum of 251 rows, including the first row with the names of the fields. If more than 250 objects are alerted, a brief warning at the end of the email indicates that only partial results are shown.

Compressed CSV attachment

Although it is possible to parse the HTML results table for integrating its data into external software, the HTML tables of Nexthink alerts were mostly designed to be read by human beings. In addition to the HTML results table, however, the email sent by investigation-based alerts includes a compressed text attachment which is much more interesting for integration purposes. The attachment is a CSV file compressed with the well known Lempel-Ziv LZ77 algorithm whose name is always set to be "alert.zip". When uncompressed, the name of the file becomes "alert.csv". This attachment holds the same data as the HTML results table, with the advantage that its contents are easier to parse.

Once the attachment is uncompressed, the resulting CSV can be easily imported into third-party tools such as your favourite spreadsheet program.

Syslog integration

The system logging service, or syslog for short, is an alternative to email for integrating data coming from Nexthink alerts. Applications typically use the syslog to store messages that keep track of the activity of the application itself or that describe a situation that the application considers relevant. The syslog service is responsible for receiving these messages, assigning them a time-stamp and storing them in a timely manner.

In the Finder, you can select to send the results of a global investigation-based alert to the system log. Please note however that only those accounts with the right permissions are able to create global alerts.

Syslog configuration

The Nexthink appliance relies on the rsyslog package for writing to the system log. Many Linux distributions use rsyslog as their default service for system logging. If you are familiar with the configuration files of rsyslog, you may modify the format of alerts and of the Engine logs in general. The format of the configuration files of rsyslog is backwards compatible with the original syslog daemon. From this point on, we may refer to rsyslog as syslog when we talk about the service itself and not about a specific feature of rsyslog. The configuration file for rsyslog is found in /etc/rsyslog.conf. For the sake of clarity, the specific modifications of the Engine to the syslog configuration are stored in a separate file which is found in /etc/nexthink/nx_rsyslog.conf. This file is applied to the main configuration file by means of an include directive in /etc/rsyslog.conf. The part of the syslog configuration file /etc/nexthink/nx_rsyslog.conf which is relevant for alerts is shown below:

```
 $template
RFC5424format,"<%pri%>1 %timestamp:::date-rfc3339% %hostname%
 %programname% %procid%%msg%\n"
 ...
 # alerts
 local5.=notice -/var/log/nexthink/alert.log;
 ...
 # alerts
 local6.=notice -/var/log/nexthink/alert.log; RFC5424format
```

The first line defines an output format for syslog messages by means of a template. The template is named *RFC5424format* because it follows the recommended format for syslog messages which is described in the most recent

Internet standard about the syslog protocol: RFC 5424. The template defines the output to be composed of a priority number followed by the timestamp, the host name, the program name, the id of the process which issued the syslog message and the message itself. Once defined in this way, a template can be applied to one or several message filters. For alerts, you can see that we declare two filters in the syslog configuration file, depending on the facility specified to log the alert. Both filters are instructed to write their output to the same file: /var/log/nexthink/alert.log. The minus sign before the file name is there to improve the performance of the syslog daemon. It indicates that syslog output to the file is buffered, so the syslog system will not directly write to the filesystem but to a buffer in memory and then really write to the disk once the buffer is full. The two filters however accept messages from different facilities. If the facility used is local5, rsyslog will use the default syslog output format. On the other hand, if the facility used is local6, rsyslog will use the output format defined by the template ?RFC5424format? for every logged alert.

Alert format

We have seen that the format of an alert in the system log depends on the facility used to log the alert: local5 for default format and local6 for RFC 5424 format. The format of the message itself also depends on which facility is used by the Engine to log the alert. You can control the facility used to log alerts by means of a configuration parameter in the engine called legacy_alert_format in the syslog tag of the configuration file:

```
<syslog>
   <legacy_alert_format>true</legacy_alert_format>
</syslog>
```

By default, the parameter is set to true in order to use the traditional alert format for syslog. Facility local5 is used in this default case. When local5 is used, the result of an alert is divided into two types of messages. The format of the first message is composed of the name of the alert and the number of rows that follow:

alert [n]

Then each row of the result is given in the following format:

alert | value1 | value2 | ? |

where alert is again the name of the alert as saved with the Finder and valueN is

the value that corresponds to the Nth field of the investigation associated to the alert. The messages are preceded by the timestamp and the default values set by syslog that depend on the default syslog configuration.

Example:

```
<default syslog prefix> Last IP alert [1]
<default syslog prefix> Last IP alert |QAXPRG|192.168.0.44|
```

You may edit the file /var/nexthink/engine/<engine_instance>/etc/nxengine.xml manually to set the value of legacy alert format to false. If the value of this parameter is set to false, facility local6 is used for logging Engine messages. When local6 is used, the message generated for an alert combined with the template defined in the syslog configuration file has the following output format:

<pri>version timestamp hostname NX pid object [engine *(field="value")] alert [number/total]

where

- **pri**: Priority of message. It is computed by first multiplying the number of the facility that sent the message by 8 and then adding the severity. The severity used by all log messages in the Engine is notice (5). Since the facility used is local6 (22) for non-legacy alerts, the priority is <181>.
- **version**: Version of syslog protocol. We use version 1.
- **timestamp**: High precision timestamp derived from RFC 3339.
- **hostname**: Qualified name of the machine at the origin of the log message.
- **NX**: This fixed value is the application name for the NEXThink Engine.
- **pid**: Process ID of the Engine in the host machine.
- **object**: Object category of the alarm investigation (e.g. source, user, destination, etc).
- **engine**: Name given to the Engine in the server tag of the configuration file. **Warning**: this is not a valid SD-ID according to RFC 5424. We use it as a convention, but it may change in the future.
- **field**: Name of the object parameter to display.
- **value**: Value of the object parameter. The list of values is the actual result of the investigation.
- **alert**: Name of the alert as saved with the Finder.
- **number/total**: Number of the current row out of the total number of rows in the investigation result.

Example: `<181>1 2011-04-15T16:56:30.966693+02:00 Barahona NX 3286 source [DebugEngine name="QAXPRG" last_ip_address="192.168.0.44"] Last IP alert [1/1]`

Known Limitations

In non-legacy alerts mode, the names of fields in the message of the logged alerts may not exactly match the names of the fields which where specified in the Finder when defining the alert. This is because the names used when generating the alert are the internal names of the fields as declared in the code of the Engine and not the names that you can see in the Finder. Usually, the two names are very similar if not equal, but do not rely blindly on Finder names to parse alert results in the system log. The result of a periodic alert in the syslog does not specify the period for which the alert has been computed. Although the timestamps can give you a hint on this period, they do not provide a definitive answer.

The operations described in this article should only be performed by a Nexthink Engineer or a Nexthink Certified Partner.

If you need help or assistance, please contact your Nexthink Certified Partner.

Related tasks

- Receiving alerts
- Creating an investigation-based alert
- Configuring the system log

Downloads

- Download the examples from the previous chapters here.

- Get the Integration Technical Presentation from here. There is an overview of why to integrate, integration hooks, success stories and questions and answers.

Web API V2 and NXQL

Introducing the Web API V2

The Web API V2 is an HTTPS service that you invoke by issuing a POST or GET HTTP request to the Engine via the URL:

```
https://<Engine IP address or DNS name>:<Web API port number>/2/query
```

The service answers NXQL queries to the in-memory Engine database with a list of records in the selected output format. By default, the Web API port number is 1671.

A request expects the following parameters:

query
: The NXQL query to execute.

platform
: Specifies the target platform of the query. Should the query target multiple platforms, supply the argument for as many platforms as required. Supported platforms are **windows**, **mac_os** and **mobile**.

format
: The expected output format. Available formats are **csv**, **html**, **xml** and **json**.

hr
: Optional: Boolean value that indicates whether the output should be human readable. When true, numerical values in the response are adapted to their best fitting units for better readability. The chosen units are also displayed along with the values. Not used in the JSON output format.

For instance, to execute the following NXQL query: `(select (device_uid name) (from device))`

Use the following Web API request. Note that the query must be URL-encoded:
`https://192.168.2.3:1671/2/query?platform=windows&platform=mac_os&query=(select%20(devi device))&format=csv`

The Engine returns the list of unique identifiers and names of all Windows and Mac OS devices in CSV format.

Template Parameters

Extra parameters **p1**, **p2**, etc. can be added to the query to replace placeholders **%1**, **%2**, etc. in the NXQL query. Use placeholders in place of the names of custom fields, names of categories or literal values for parameterizing queries that are used often.

For instance, the following NXQL query to look returns the name of all devices, as well as their associated keyword from a category that you pass as a parameter `(select (name #%1) (from device))`

Use the following Web API request to get the names of all devices and their *Location* keyword:

```
https://<engine>:1671/2/query?query=(select%20(name%20%23%251)%20(from%20device))&p1=Lo
```

Authentication

Any account with Data Privacy set to **none (full access)** and the option **Finder access** enabled can make use of the Web API. However only those with the right to edit categories can perform updates.

User credentials are verified with basic HTTP authentication. For a given user, the visibility and info levels are identical to those defined in their profile in the Portal.

HTTP Status Codes

The Web API V2 returns:

- **200 OK**: If the request is successful;
- **400 Bad Request**: If the request is invalid;
- **401 Not Authorized**: If no credentials are provided or if they are not valid;
- **403 forbidden**: If Web API is not licensed.

Testing the Web API V2 with the NXQL editor

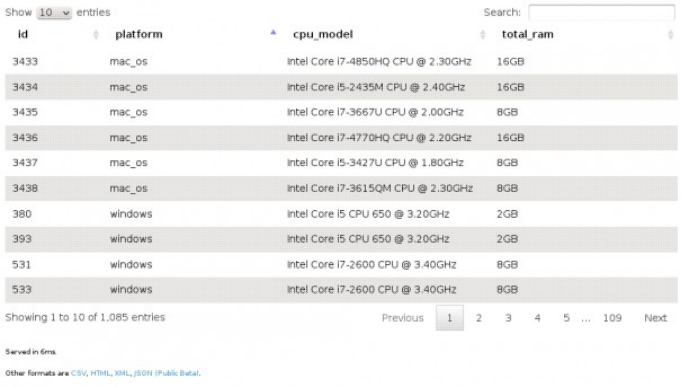
The NXQL editor is a web-based user interface to the Web API V2. This useful editor lets you test the queries that you will later use in your integration projects. The NXQL editor is present in every Engine with the Integration tookit and you can access it from your favorite web browser by typing in the following URL:

```
https://<Engine IP address or DNS name>:<Web API port
number>/2/editor/nxql_editor.html
```

To write a query in the NXQL editor:

1. Provide the user credentials. Type in the user name and password in the two text input boxes at the top. The access rights of the user associated to the supplied credentials apply to the query.
2. Select the platforms that your query targets by ticking the appropriate platform icons at the top right corner.
3. Type in your NXQL query inside the big text region in the middle.
    - ♦ If your query includes any placeholder for template parameters, specify the value of the parameters in the two text boxes below the query. Editor queries may include up to two template parameters.
4. Optional: Tick **Formatted** to get a human readable output (see **hr** parameter of Web API V2 requests above).
5. Click **Send**.

Once you send your query, the editor displays the message **Loading...** while the Engine is processing it. After a few seconds, depending on the speed of your connection, the complexity of your query and the load on the Engine, the response appears below the **Send** button in the same page of the NXQL editor:



- Choose the maximum number of displayed rows with the **Show x entries** picker.
- Navigate through the result pages with the help of the buttons at the bottom right.
- Order the results by column in ascending or descending order by repeatedly clicking the title of the column.
- Click the **Other formats** options at the bottom left to get the results in CSV, HTML, XML or JSON format.

Using the Web API V2 with wget

The Web API V2 can easily be invoked using the classic UNIX tool **wget**. For instance, to retrieve the names of all devices in CSV format using **wget**, write the following command:

```
> wget --quiet \
  --no-check-certificate \
  --user=admin --password=admin \
  --output-document devices.csv \
  'https://our-engine-dns-name:1671/2/query?
   query=(select%20(id%20name)(from%20device))%20&
   format=csv&
   platform=windows&platform=mac_os'
```

Using the Web API V2 with PowerShell

The Web API can be invoked using Windows PowerShell, however, since the standard Invoke-WebRequest CmdLet does not support self-signed certificate, you should use the CmdLet defined in the downloadable file Code-For-Invoke-Nxql.ps1. After saving this script, load it into your PowerShell environment. Make sure that your PowerShell execution policy is set to *unrestricted*.

To load the script, type in the following in the PowerShell console:

```
. ./Code-For-Invoke-Nxql.ps1
```

To retrieve the list of names of all the devices of any platform in CSV format, for example, execute the following command:

```
Invoke-Nxql -ServerName 192.168.2.3
  -UserName admin -UserPassword admin
  -Platform windows,mac_os
  -Query "(select (name) (from device))" > devices.csv
```

To get the full command line options, type in:

```
Invoke-Nxql -?
```

Using the Web API V2 with Power BI Desktop

Microsoft Power BI is a suite of interactive visualization tools that deliver analytical insights throughout your organization. Thanks to the Web API, feed Power BI with Nexthink data and graph the digital experience of end users on visually appealing charts.

To invoke the Web API V2 from Power BI Desktop:

1. Open Power BI Desktop.
2. Select **Get Data** from the top menu.
    1. Choose **Web** as data source.
    2. In the **From web** dialog, type in your Web API query as **URL**, requesting XML as output format.
        - ◊ Remember that you can easily get this URL while editing your query in the NXQL editor by copying the link from the **Other formats** section at the bottom of the page.
    3. Click **OK**.
3. Type in your Web API credentials when prompted to do so.
4. In the Navigator, select the **body** to get the XML data from the Engine response.
5. Click **Load**.

The data source is now added. However, Power BI is unable to send authentication credentials again when using these parameterized queries. The solution is to divide the query into a base and a relative path:

1. In Power BI Desktop, select **Edit Queries** from the top menu.
2. Open the data source that calls the Web API and select **Advanced Editor**.
3. In the editor, look for the **Source** variable under the **let** keyword:
    ```
    Source=XML.Table(Web.Contents("https://our-engine-dns-name:1671/2/query?
    query=(select%20(id%20name)(from%20device))%20&format=xml"
    ...
    ```
4. Divide the URL of the query into a base and a relative path as follows:
    ```
    Source=XML.Table(Web.Contents("https://our-engine-dns-name:1671/2/",
    [
    RelativePath="query?query=(select%20(id%20name)(from%20device))%20&format=x
    ]...
    ```

For each subsequent query, Power BI Desktop will use the credentials that you typed in when you first created the data source.

Related concepts

- Platform

NXQL Tutorial

Overview

The Nexthink Query Language (NXQL) is a language designed to query the in-memory database of the Nexthink Engine via the Web API V2. The language is loosely based on SQL, using similar keywords in its statements, but with a LISP-like syntax.

NXQL is the evolution of the *selector* language (another pseudo-SQL internally developed language). The Finder, the Portal and the Lua scripts running within the embedded Lua interpreter of the Engine currently use the selector language to query the Engine. Being specifically designed for integrations and with speed improvements in mind, NXQL outperforms the selector language in many areas. NXQL lets you write more complex queries and, since you have more control over the object traversal, queries typically execute faster.

This tutorial is meant to guide you through the process of learning NXQL by example. Follow the NXQL tutorial in the suggested order to get the most out of it.

To execute the queries in the tutorial, use the NXQL editor that is available in every Engine with the Integration toolkit module. The rest of the tutorial assumes that you are authenticated in the NXQL editor with admin credentials, so you have the access rights to see all available data (such as the name of computers and users).

First queries

To get a list with the unique identifiers and the names of all available devices, enter the following query:

```
(select (device_uid name) (from device))
```

Note that the query starts with an opening parenthesis and ends with a closing parenthesis. The number of opening and closing parentheses must be balanced for the query to be well formed. To help you formulate your queries, the system automatically adds missing parentheses at the end when needed. The query starts with the keyword **select** and it is thereby called a select statement. The select statement includes a list of the fields to be retrieved and a **from** clause that specifies the table where the fields are found.

```
(select            - select statement
   (id name)       - list of fields
   (from device))  - queried table
```

Within a query, fields may contain wildcard characters. For instance, to get the names and all the antivirus related fields of devices, type in the following query:

```
(select (name *antivirus*) (from device))
```

If you mistype the name of a field, the system signals the error and suggest as alternative either the exact name of the field that you most probably mispelled or, if no field exists whose name is close enough to the input, the complete list of field names that you can use in that context.

To retrieve only a subset of the devices, filter the results by the value of some of the fields. For example, to select the device named **NXT-DV10** only, type in the following query:

```
(select (name)
   (from device
   (where device
       (eq name (string "NXT-DV10")))))
```

Inside the **from** clause, the **where** clause keeps only those devices whose name is equal to NXT-DV10. The first argument of a **where** clause is the table to which the filter applies, and the second argument is the expression of the filter itself. A filter is composed of an operation, followed by the name of a field and a typed value. The possible operations are **eq**, **ne**, **lt**, **le**, **gt** and **ge** meaning equal, not equal, less than, less or equal, greater than, and greater or equal, respectively. The type of the value that must match the type of the field. Find the names and the types of all the fields in the data model.

Logical-and operation

You can define a **where** clause for more than one filter. In this case, only those objects matching all the filters are selected.

For instance, the following query returns the list of all devices running Windows 7 with no antivirus installed:

```
(select (name os_version_and_architecture number_of_antiviruses)
   (from device
   (where device
       (eq os_version_and_architecture (pattern "Windows 7*"))
       (eq number_of_antiviruses (enum 0)))))
```

Logical-or operation

On the other hand, if you want to retrieve objects that either match one set of filters or another, you have to write two **where** clauses for the same kind of object.

For instance, to retrieve the list of devices running Windows 7 or Windows 8 / 8.1, type the following query:

```
(select (name os_version_and_architecture number_of_antiviruses)
   (from device
   (where device
           (eq os_version_and_architecture (pattern "Windows 7*")))
   (where device
           (eq os_version_and_architecture (pattern "Windows 8*")))))
```

Remember that this is valid for *where* clauses on the **same kind of object** only. When writing more advanced queries that set conditions on objects of different tables, keep in mind that multiple *where* clauses on different kinds of objects behave as a logical-and. Examples will follow below.

At this stage, you are already able to query any field of any object tables defined by Nexthink. You may try with other objects different from device, such as user or binary, to get more familiar with the NXQL.

Using Events

An event is an occurrence in your IT infrastruture that happens at a defined moment in time. All events have a timestamp, therefore events can be ordered by time. Events are at the core of Nexthink technology, being the basic information units of the in-memory database. Depending on the kind of occurrence that they describe, there are several types of events. Each type of event is linked to a well-defined set of objects. For instance, **connection** events are linked to **user**, **device**, **binary**, **destination**, and **port** objects.

The number of events in the database is usually several orders of magnitude higher than the number of any other kind of object. While an object table like the device table may contain from a few hundreds to ten thousand elements, the event table may hold tens of millions of elements. For performance reasons, it is important to keep this in mind when setting the time span of a query involving events.

In your queries, you can use the event table in two ways:

- Directly selecting those events that occur during a given time interval. For instance, to retrieve the last 100 connection made by **firefox.exe** the last day:

```
(select (start_time end_time incoming_traffic outgoing_traffic)
   (from connection
       (where binary (eq executable_name (pattern firefox.exe)))
       (between midnight-1d midnight))
   (limit 100)
   (order_by start_time desc))
```

- Selecting those objects that are linked to events occurring during a given time interval. For instance, retrieve all devices that used **firefox.exe** to access the web yesterday:

```
(select (id name)
   (from device
       (with connection
           (where binary (eq executable_name (pattern firefox.exe)))
           (between midnight-1d midnight))))
```

While the former query is similar to queries made so far, the latter introduces the **with** clause. This clause specifies the type of events to traverse in order to build the list of selected objects. Of course, only those events that are linked to the object of interest can be used for the traversal.

In addition to events, the **with** clause can also precede the **package** keyword when it expresses the relationship between a device and a package object, as explained below.

Logical operation with events

You can refine your query even further. Let us suppose that you are interested in those devices using **firefox.exe** that accessed **mail.google.com** yesterday:

```
(select (id name)
   (from device
       (with web_request
           (where binary (eq executable_name (pattern firefox.exe)))
           (where domain (eq name (string mail.google.com)))
        (between midnight-1d midnight))))
```

Note that the query holds two *where* clauses which apply to two different kinds of objects: binary and domain. Thus, they behave as a logical-and, meaning that the two conditions must be satisfied.

To behave as a logical-or, the *where* clauses must apply to the same kind of object. For example, to expand our query to those devices that used **chrome.exe** in addition to **firefox.exe** for accessing **mail.google.com** yesterday, write:

```
(select (id name)
  (from device
      (with web_request
          (where binary (eq executable_name (pattern firefox.exe)))
          (where binary (eq executable_name (pattern chrome.exe)))
          (where domain (eq name (string mail.google.com)))
       (between midnight-1d midnight))))
```

On the other hand, to refine our original query even more and return only those devices which used a version of **firefox.exe** lower than **50**, type in:

```
(select (id name)
  (from device
      (with web_request
          (where binary (eq executable_name (pattern firefox.exe))
                        (lt version (pattern 50)))
          (where domain (eq name (string mail.google.com)))
       (between midnight-1d midnight))))
```

That is, set several conditions on the *where* clause of the same kind of object (the **binary** object, in this case) for the conditions to be combined with a logical-and.

Finally, in the rarer cases where you need to combine conditions on different kinds of objects with a logical-or, use the **union** keyword documented below.

Computing aggregates

The selection of objects linked to events can be augmented with *aggregates*. An aggregate is a named function that computes a count, a sum or an average of a given field for all selected events. For instance, the **incoming_traffic** aggregate

adds up all the values of the field **incoming_traffic** of all the **connection** or **web_request** events selected by a **with** clause. Specify aggregates in a **compute** clause inside a **with** clause.

Since some aggregates require the traversal of events for their computation, you have similar performance concerns when using aggregates as when using events in your queries. It is important to limit the time interval of queries that may otherwise need to traverse many millions of events. Thus, aggregates which are not marked as **FP** in the data model require a **between** clause to limit the traversal. The **between** clause, however, does not put a strict limit on the time interval that you can specify. It is your responsibility to set a reasonable time interval, especially if the query is going to be periodically repeated.

For instance, to compute the incoming traffic per device of all web requests made to **mail.google.com** during the last 7 days, write the following query:

```
(select (id name)
   (from device
       (with web_request
       (where domain
           (eq name (string mail.google.com)))
       (compute incoming_traffic)
       (between midnight-7d midnight))))
```

The list of aggregates for each event table is defined in the NXQL data model.

At this stage, you may wonder how to filter devices based on the value of an aggregate. In our previous example, you may want to select devices which transferred 1GB of data yesterday. This is the purpose of the **having** clause, which may appear in a **from** clause within a **with** clause. Of course, the aggregates filtered by the **having** clause must be declared first inside the **compute** clause.

```
(select (id name)
   (from device
       (with web_request
           (where domain
               (eq name
                   (string mail.google.com)))
           (compute incoming_traffic)
           (between midnight-7d midnight))
   (having
       (gt incoming_traffic
           (byte 1073741824))))
```

Using categories and custom fields

In NXQL, both categories and custom fields are treated equally. They behave like classic fields, but their name is prefixed by the **#** character. For instance, to retrieve the list of devices with their **Location**, given that Location is a category on device, write the following query:

```
(select (id name #Location) (from device))
```

You can also use categories or custom fields as filters:

```
(select (id name)
   (from device
       (where device
           (eq #Location (enum Paris)))))
```

The names of categories or custom fields containing spaces or quotes must be quoted:

```
(select (id name)
   (from device
       (where device
           (eq #"My Location" (enum Paris)))))
```

Campaigns custom fields

The results of campaigns are visible in NXQL as custom fields of the object **user**. The name of custom fields related to campaigns have the following format:

```
#"campaign:Name of the campaign/Name of the question"
```

Note the use of the keyword **campaign:** at the beginning of the name of the custom field. For example, to know the answers of every user to the question **Device preference** within the campaign **Laptop satisfaction**, write the query:

```
(select (name #"campaign:Laptop satisfaction/Device preference")
   (from user))
```

The underlying type of an answer to a *single answer* or *opinion scale* question is the **string** type. In turn, the underlying type of an answer to a *multiple answer* question is a list of strings. Compare the values of an answer with the **eq** and **ne** operators (no other operator is allowed for comparing answer values). For example, to get the name and the actual answer of all the users who did not answer **No** to the single answer question **Device preference**, write the query:

```
(select (name #"campaign:Laptop satisfaction/Device preference")
   (from user
       (where user (ne #"campaign:Laptop satisfaction/Device
preference"
                   (string "No")))))
```

Similarly, to select the users who did not answer a specific single answer or opinion scale question yet, compare with the empty string:

```
(select (name #"campaign:Laptop satisfaction/Device preference")
   (from user
       (where user (eq #"campaign:Laptop satisfaction/Device
preference"
                   (string "")))))
```

In the case of multiple answer questions, it is possible to query for combinations of answers in the response given by the users. Use the logical-and and logical-or operations in the **where** clause described above or specify a list of values to exactly match a particular combination. For example, to get the users who answered both **Speed** and **Size** (and possibly something else) to the **Positive points** question of the campaign **Laptop satisfaction**, write the query:

```
(select (name #"campaign:Laptop satisfaction/Positive points")
   (from user
       (where user (eq #"campaign:Laptop satisfaction/Positive points"
                   (string "Speed"))
                   (eq #"campaign:Laptop satisfaction/Positive points"
                   (string "Size")))))
```

Instead, if you want to query for the users that exactly answered **Speed** and **Size** and nothing else, specify them as a list:

```
(select (name #"campaign:Laptop satisfaction/Positive points")
   (from user
       (where user (eq #"campaign:Laptop satisfaction/Positive points"
                   (list ("Speed" "Size"))))))
```

Alternatively, to get the users that chose one of the values **Speed** or **Size** (or both), write the logical-or version of the query:

```
(select (name #"campaign:Laptop satisfaction/Positive points")
   (from user
       (where user (eq #"campaign:Laptop satisfaction/Positive points"
                   (string "Speed")))
       (where user (eq #"campaign:Laptop satisfaction/Positive points"
                   (string "Size")))))
```

Finally, to get the users that did not give any answer yet to a multiple answer question, compare with the **nil** keyword instead of an empty string:

```
(select (name #"campaign:Laptop satisfaction/Positive points")
   (from user
       (where user (eq #"campaign:Laptop satisfaction/Positive points"
nil))))
```

Scores custom fields

Scores are accessible through NXQL as special custom fields of the objects **device** or **user**. The name of custom fields related to scores have the following format:

```
      #"score:Name of the score definition/Name of the score"
```

Note the use of the keyword **score** at the beginning of the name of the custom field. For example, to get the **Boot speed** leaf score of all devices, which is inside the **Device performance** score definition, write the query:

```
(select (name #"score:Device performance/Boot speed")
   (from device))
```

Because scores hold numerical values, the underlying type of any score is the **real** type. As an example of putting a condition on the value of a score, the following query retrieves all the devices whose **Boot speed** score is higher than 5.0:

```
(select (name #"score:Device performance/Boot speed")
   (from device
       (where device (gt #"score:Device performance/Boot speed"
                     (real 5.0)))))
```

Apart from numerical values, a score may have no value at all. To query for objects with an empty score, compare the value of the score with the **nil** keyword using the **eq** or **ne** operators. For example:

```
(select (name #"score:Device performance/Boot speed")
   (from device
       (where device (eq #"score:Device performance/Boot speed"
                      nil))))
```

Using platforms

NXQL supports the three platforms included from Nexthink V5.3: Windows, Mac, and Mobile.

- When using the NXQL editor, select the platforms to which the query applies by ticking the check boxes at the top right corner of the editor.
- When directly querying the API via an HTTP request (e.g. from a script or an integration), use the **platform** parameter described in the introduction.

When selecting multiple platforms, beware that only those tables and fields that are common to all the selected platforms are valid in your query. For instance, the field **name** of a device is available for all three platforms, but **all_antiviruses** is available only for devices of the Windows platform. Therefore, a multi-platform query that includes the field **all_antiviruses** is not valid.

Selecting multiple tables

There are two types of queries in NXQL which let you combine information from multiple tables:

- Selecting unique pairs of objects in relation to events of a particular kind.
- Selecting events of a particular kind, as well as information from objects linked to those events.

Although they may look similar, both types of queries differ in some aspects that we detail below.

The most common type of query that requires multiple tables is the selection of unique pairs of objects which took part in a series of events. In this type of query, you can select only two object tables, while you specify the event table that makes the link between each pair of objects inside a **with** clause. In the **select** clause, specify the name of each object table before its corresponding list of fields, and then repeat the names of the object tables in the **from** clause. For instance, if you are interested in the names of both the users that executed **firefox.exe** and the devices on which it was executed, write the following query:

```
(select ((device name) (user name))
   (from (device user)
       (with execution
           (where binary
               (eq executable_name (pattern firefox.exe)))))
   (limit 100))
```

In the second type of query, the main interest lies in the individual events of the selected event table, which you may decorate with information from the objects linked to each event. Thus, to write queries of the second type, specify the name of the event table and the names of each additional object table in the **from** clause, as well as before each corresponding list of fields of interest in the **select** clause. For example, the following query returns the last 100 connections of **firefox.exe**, as well as the names of the devices that originated each connection:

```
(select ((device (name))
         (connection (start_time end_time incoming_traffic
outgoing_traffic)))
   (from (device connection)
       (where binary (eq executable_name (pattern firefox.exe))))
   (limit 100)
   (order_by start_time desc))
```

In this second type of query, objects may be repeated in the results if they are linked to multiple events. For instance, in the example above, there may be a device which is linked to more than one of the selected connections. The name of that device will therefore appear repeated for each related connection. That is the opposite of the first type of query, where you get unique pairs of objects which may be linked to many events and you are not interested in the individual events.

Despite the given example, you may have noticed that queries of the second type are not limited to two tables. You must select one event table and one or more object tables instead. For example, to get all the executions of binaries that do not have their threat level set which took place today and display their binary path, along with some info about the binaries, devices, and users involved, write:

```
(select
    (
        (execution binary_path)
        (binary (executable_name version))
        (device (name last_ip_address))
        (user (name))
    )
    (from (execution binary device user)
        (where binary (eq threat_level (enum "-")))
        (between midnight now)
    )
    (limit 100))
```

As for constraints, both types of multiple table queries require a **limit** clause to restrict the maximum number of returned entries and they do not allow the

computation of aggregates.

Using packages in queries

Package is a special keyword in NXQL in the sense that it can function as an object table or as a relationship table. Indeed, a package can refer to an installed package itself, with its attributes such as name, version, company, etc. or to its relation with devices through its installation. That is the reason why you can use packages inside a **with** clause, which is otherwise reserved to events.

For instance, to list all devices with the package **Microsoft Office 365** installed, write the following query (package works as relation):

```
(select (name)
   (from device
       (with package
           (where package (eq name (pattern "Microsoft Office 365
ProPlus*")))))
```

To get the package version along with the device, write the following query (where package works both as object and as relation):

```
(select ((device (name)) (package (version name publisher)))
   (from (device package)
       (with package
            (where package
               (eq name (pattern "Microsoft Office 365 ProPlus*"))
               (eq type (enum program)))))
   (limit 10000))
```

If you simply want to compute the number of packages installed on every device, write the following query (where package works as relation):

```
(select (name)
   (from device
       (with package
           (compute number_of_packages)))
```

Operations on sets of objects

With NXQL, it is possible to compute two lists of objects of the same type and combine them into a single result with just one query.

For example, to compute the list of devices without the package **Microsoft Office**:

```
(select (name)
   (except
       (from device)   - list of all devices
       (from device    - list of device with Office
           (with package
               (where package (eq name (pattern
*Microsoft*Office*)))))))
```

To execute the query above, the system computes the list of all devices and subtracts from it the list of devices with **Microsoft Office**, creating logically the list of devices without **Microsoft Office**.

Three set operators exists:

- **except** (A) (B): Return objects appearing in A but not in B.
- **union** (A) (B): Return all objects appearing in A or in B.
- **intersect** (A) (B): Return only those objects appearing both in A and in B.

Remember that only one object table can be used in the two **from** clauses linked by a set operator. It is impossible to do an union of devices and users, for instance.

Note as well that these operators work with object tables only and not with event tables.

Updating values of categories and custom fields

To update a dynamic field, i.e. a category, use an **update** statement. An **update** statement sets the values of the specified dynamic fields in all the objects selected by a **from** clause. For instance, to set the location of some devices to Paris, based on their last IP address, write the following query:

```
(update (set #Location (enum Paris))
   (from device
       (where device
           (eq last_ip_address (ip_network 172.16.12.0/16)))))
```

Setting category overrides the auto-tagging rules associated with an keyword. If you want to reactivate the auto-tagging rules, write the following query.

```
(update (set #Location nil)
   (from device
       (where device
           (eq last_ip_address (ip_network 172.16.12.0/16)))))
```

Note that the table returned by an **update** statement contains the identifiers of all modified objects

Using placeholders

To generalize a query that you execute often, use placeholders. A placeholder is a number prefixed by the **%** character that you put in the place of a value, or a custom field name, or a category name inside a query. When the query is executed, each placeholder is replaced by the actual value supplied as parameter. For example, the following query includes two placeholders:

```
(select (id name)
   (from device
       (with web_request
           (where device (eq #%1 (enum %2)))
           (between midnight-1d midnight))))
```

To execute this query, you should provide the name of a custom field or category for devices and its actual value as parameters. In the NXQL editor, provide the parameter values in the two text boxes for parameter input below the query.

In programmed queries, provide the actual parameters in the HTTP request.

NXQL language definition

While the NXQL tutorial guides you through your first steps with NXQL, this document gives a more formal definition of the query capabilities of NXQL.

Selecting plain objects

To select objects from an object table, use this form of the select statement:

```
(select ([field]...)
    (from [object]
        (where [object] [filter])...))
```

Example:

```
(select (device_uid name)
    (from device))
```

Selecting plain events

To select events from an event table, use this form of the select statement:

```
(select ([field]...)
    (from [event]
        (where [event] [filter]...)...
        (between datetime datetime))
    (order_by start_time [asc|desc]) // optional
    (limit number))
```

Example:

```
(select (start_time incoming_traffic outgoing_traffic)
    (from connection
        (where connection (ne status (enum established))
                          (ne status (enum closed)))
        (where user (eq name (string "siesme@AONNETWORK")))
        (between now-7d now))
    (order_by start_time asc)
    (limit 100))
```

This query returns the start time and the incoming and outgoing traffic of the last 100 connections whose status is not equal to **established** or **closed**. That is, those connection with a status equal to **rejected**, **no host** or **no service**.

Selecting events with decoration

To select events and their linked objects from a given event table, use the following form of the select statement. Note that there is no limit on the number of object tables that you can specify, as long as the object table is really linked to the events. For instance, it would not make much sense to query about printers related to execution events, since printers are not linked to executions.

```
(select (([object|event] [field]...)...)
    (from ([event] [object]...)
        (where [object|event] [filter]...)...
        (between datetime datetime))
    (order_by start_time [asc|desc]) // optional
    (limit number))
```

Example:

```
(select ((connection (start_time)) (user (name)))
    (from (connection user)
        (where connection (ne status (enum established))
                          (ne status (enum closed)))
        (between now-7d now))
    (order_by start_time desc)
    (limit 100))
```

The query returns the start time as well as the name of the user who initiated the last 100 connections whose status is not equal to **established** or **closed**, that is, with a status equal to **rejected**, **no host** or **no service**.

Another example:

```
(select ((user (name)) (device (name)))
    (from (connection user device)
        (where connection (ne status (enum established))
                          (ne status (enum closed)))
        (between now-7d now))
    (order_by start_time desc)
    (limit 100))
```

This last query is identical to the previous one, except for that it does not return the start time of the connection. Since these kind of queries return one tuple per event, you may see a tuple with the same user name and device name appearing more than once in the results. These tuples are not really duplicated results, they actually belong to different connections although you may not see the difference due to the selected fields.

Selecting objects with activity

To select objects linked to an activity (event), use the following select statement. The difference with the previous family of queries is that in the former you get one result tuple per event, while in this latter you get one result tuple per object.

```
(select ([field]...)
    (from [object]
        (with [event]
            (where [object|event] [filter]...)...
            (compute [aggregate]...)      // optional
            (between datetime datetime))
        (having [filter on aggregate]...) // optional
    (order_by [field] [asc|desc])         // optional
    (limit number))                       // optional
```

Example:

```
(select (name)
    (from device
        (with execution
            (where binary (eq threat_level (enum high)))
            (where binary (eq threat_level (enum intermediate)))
            (compute number_of_binaries)
            (between midnight-1d midnight)))
   (limit 100)
   (order_by name desc))
```

This query returns those devices which executed a binary whose threat level is **intermediate** or **high** yesterday. In addition, for each device, the query computes the number of distinct binaries matching the condition.

Selecting two objects

To select unique pairs of objects linked to a given type of events, use the following select statement. Note that you can select no more than two object tables and that you cannot use any logic operator.

```
(select (([object] [field]...)...)
    (from ([object] [object])
        (with [event]
            (where [object|event] [filter]...)...
            (between datetime datetime))
    (limit number))
```

Example:

```
(select ((package name) (device name))
    (from (package device)
        (with package
            (where package (eq name (pattern "*Office*")))))
    (limit 100))
```

This query returns the unique pairs of devices and packages, where the name of the package contains the term **Office**.

Updating objects

The update statement modifies categories or custom fields of an object table:

```
(update (set [field] ([type] [value]))...
    (from [object]
            (where [object] [filter]...)...))
```

To reset the value of a category or custom field, use the following update statement:

```
(update (set [field] nil)...
    (from [object]
            (where [object] [filter]...)...))
```

Examples:

```
(update (set #Location (enum Paris))
    (from device
        (where device (eq name (pattern "PA*")))))
```

This query updates the **Location** category of every device whose name begins with **PA** to **Paris**.

```
(update (set #Location nil)
    (from device
        (where device (eq name (pattern "PA*")))))
```

This query resets the **Location** category to *nil*. If an auto-tagging rule for the **Location** of devices is in force, the system will reset the value to the keyword of the matching auto-tagging rule.

Filter

A filter is condition on a field value. It has the following format:

```
([comparer] [field] ([type] [value]))
([comparer] [field] nil)
```

Where [comparer] may have one of the following values:

- **eq**: equal. If the type of the field is an array of [type], **eq** is true if at least one element of the array is equal to the value.
- **ne**: not equal. If the type of the field is an array of [type], **ne** is true if no element of the array is equal to the value.
- **lt**: less than.
- **le**: less or equal.
- **gt**: greater than.
- **ge**: greater or equal.

Where [type] may have one of the following values:

- **boolean**: A true or false value. Use keywords *true* and *false*, *yes* and *no*, or *1* and *0* as boolean literals.
- **string**: A string, If the string contains a space or a double-quote, it must be double-quoted and the quote duplicated, e.g "Softy ""Visual""".
- **integer**: An integer number, e.g. 10.
- **real**: A floating-point number, e.g. 12.56.
- **enum**: A list of distinct values. As in the case of strings, if the value contains a space or a double-quote, it must be double-quoted.
- **second**: A natural number representing seconds, e.g. 60 second (= 1 minute).
- **millisecond**: A natural number representing milliseconds, e.g. 60000 millisecond (= 1 minute).
- **microsecond**: A natural number representing microseconds, e.g. 60000000 microsecond (= 1 minute).
- **byte**: A natural number representing bytes, e.g. 1048576 byte (= 1MB).
- **ip_address**: An IP address, e.g. 172.16.10.5.
- **ip_network**: An IP network, e.g. 172.16.0.0/16.
- **mac_address**: A MAC address, e.g. 48:5b:39:18:70:bb.
- **mhz**: A natural number representing mega hertz, e.g. 1600 mhz (= 1.6 GHz).
- **sid**: A Windows security token, e.g. S-1-5-21-3623811015-3361044348-30300820-1013.
- **md5**: A MD5 hash code in hexadecimal format, e.g. d41d8cd98f00b204e9800998ecf8427e.
- **port**: A port type (udp/tcp) followed by a port number, e.g. tcp/8080.
- **version**: Four integers separated by a '.', e.g. 5.1.0.34.
- **datetime**: A date and time in ISO 8601 format, e.g. 2014-06-12T13:54:51.
- **time**: A time in ISO 8601 format, e.g. 13:54:51.
- **date**: A date in ISO 8601 format, e.g. 2014-06-12.

- **day**: A natural number representing days, e.g. 7 days (= 1 week).

Use the special type **pattern** to match a string against a star pattern expresssion. Note that only the **eq** and **ne** operators are available for the type **pattern**, for example:

```
(eq name (pattern "NY*"))
```

Filters belonging to the same **where** clause are composed with a logic *AND*. For instance, the following **where** clause selects only devices whose name begins with *NY* and whose manufacturer is Dell:

```
(where device (eq name (pattern "NY*"))
(eq device_manufacturer (string "Dell"))
```

Between

Date and time in a **between** clause is composed of a date time in ISO 8601 format or one of the following keywords:

- **now**: query time.
- **midnight**: last midnight.
- **sunday**: last Sunday at 00:00:00.
- **monday**: last Monday at 00:00:00.
- **tuesday**: last Tuesday at 00:00:00.
- **wednesday**: last Wednesday at 00:00:00.
- **thursday**: last Thursday at 00:00:00.
- **friday**: last Friday at 00:00:00.
- **saturday**: last Saturday at 00:00:00.

Optionally followed by a positive or negative integer and one of the following units:

- **w**: week i.e. 7 days.
- **d**: day i.e. 24 hours.
- **h**: 1 hours.
- **m**: 1 minutes.
- **s**: 1 second.

Examples:

- (between midnight now): today.
- (between midnight-1d midnight): yesterday.

- (between monday monday+24h): last monday.
- (between 2014-7-16@14:00:00 2014-7-16@15:00:00): on 2014-7-16 between 2 and 3 PM.

NXQL Data Model

Objects

application

An application is a sets of executables e.g. 'Microsoft Office'. Platforms:

Name	Type	Properties
company	string	
	Company producing the application	
	permill	
database_usage	Percentage of the database used by information related with the application	
description	string	
	Application description	
first_seen	datetime	NU
	First time activity of the application was recorded on any device.	
id	identifier	
	Unique application identifier	
known_packages	string	
	List of packages known to contain the application. This list is not exhaustive: The presence of a package does not necessarily imply that on a given device the application was installed through that package.	
last_seen	datetime	NU
	Last time activity of the application was recorded on any device.	
name	string	
	Application name	

	enum
platform	The platform (operating system family) on which the application is running.
	enum
storage_policy	Indicates the event storage policy for the application. Possible values are: • all: web requests, connections and executions are stored; • connections and executions; • executions; • none: no activity is recorded.
	day
total_active_days	Total number of days the application was active.

binary

A binary is an executable binary files identified by its hash code. Platforms:

Name	Type	Properties
	string	SE
application_category	Indicates the category of the application: • '-': Not yet tagged; • Unknown: Not categorized by Nexthink Library.	
application_company	string Application company	
application_name	string Application name	
architecture	enum Executable architecture (32/64 bit)	
average_cpu_usage	permill	

	Average CPU usage for the binary
average_memory_usage	byte NU Average memory usage for the binary
average_number_of_graphical_handles	integer NU Average number of graphical handles (GDI)
company	string Executable company
database_usage	permill Percentage of the database used by information related with the binary.
description	string Description as it appears in the binary file.
executable_name	string Executable name
file_size	byte Binary file size
first_seen	datetime NU First time activity of the binary was recorded on any device.
hash	md5 Hash code of the binary (MD5)
id	identifier Unique binary identifier
last_seen	datetime NU Last time activity of the binary was recorded on any device.
paths	path List of paths of the binary
platform	enum The platform (operating system family) on which the binary is running.
sha1	sha1

	SHA-1 hash code of the binary
sha256	sha256
	SHA-256 hash code of the binary
	enum
storage_policy	Event storage policy for the binary (connection and execution, execution-only or none)
	enum SE
threat_level	Indicates the threat level of the binary: • '-': Not yet tagged; • none detected: No known threat; • low: low threat; • intermediate: Intermediate threat; • high: high threat.
	day
total_active_days	Total number of days the binary was active.
	boolean
user_interface	Application has interactive user interface
version	version
	Version of the binary

destination

A destination is a device or server receiving TCP/UDP connections. Platforms:

Name	Type	Properties
	permill	
database_usage	Percentage of the database used by information related with the destination	
	datetime	NU
first_seen	First time activity to the destination was recorded on any device.	
	identifier	
id		

	Unique destination identifier
ip_address	ip_address IP address for the destination
last_seen	datetime NU Last time activity to the destination was recorded on any device.
name	string Reverse lookup name

device

A device is Windows physical or virtual machine monitored by a Nexthink Collector. Platforms:

Name	Type	Properties
administrator_account_status	enum Determines whether the local Administrator account is enabled or disabled.	
all_antispywares	string Summary information about all the detected antispyware: • unknown: Indicates that the information could not be retrieved; • N/A: This field is not available on this operating system; • '-': No data, incompatible collector version or the data is not yet available.	
all_antiviruses	string Summary information about all the detected antiviruses: • unknown: Indicates that the information could not be retrieved; • N/A: This field is not available on this operating	

system;
- '-': No data, incompatible collector version or the data is not yet available.

all_firewalls

string

Summary information about all the detected firewalls:

- unknown: Indicates that the information could not be retrieved;
- N/A: This field is not available on this operating system;
- '-': No data, incompatible collector version or the data is not yet available.

allow_non_provisionable_devices

boolean NU

Indicates whether a device which does not fully support the policy is still allowed to connect to the Exchange Exchange ActiveSync server. If 'yes', the security policy is not guaranteed to be applied, even if the field 'ActiveSync policy application status' value is 'applied in full'

antispyware_name

string NU

Name of the main antispyware

antispyware_rtp

enum

Indicates whether the antispyware real time protection (RTP) is active:

- on: Indicates that RTP is active;
- off: Indicates that either RTP is not active or no antivirus has been detected;
- unknown: Indicates that the information could not be retrieved;
- N/A: This field is not available on this operating

system;
- '-': No data, incompatible collector version or the data is not yet available.

antispyware_up_to_date

enum

Indicates whether the antispyware is up-to-date:

- yes: Indicates that antispyware is up-to-date;
- no: Indicates that either the antispyware is not up-to-date or no antispyware has been detected;
- unknown: Indicates that the information could not be retrieved;
- N/A: This field is not available on this operating system;
- '-': No data, incompatible collector version or the data is not yet available.

antivirus_name

string NU

Name of the main antivirus

antivirus_rtp

enum

Indicates whether the antivirus real time protection (RTP) is active:

- on: Indicates that RTP is active;
- off: Indicates that either RTP is not active or no antivirus has been detected;
- unknown: Indicates that the information could not be retrieved;
- N/A: This field is not available on this operating

system;
- '-': No data, incompatible collector version or the data is not yet available.

enum

Indicates whether the antivirus is up-to-date:

- yes: Indicates that antivirus is up-to-date;
- no: Indicates that either the antivirus is not up-to-date or no antivirus has been detected;
- unknown: Indicates that the information could not be retrieved;
- N/A: This field is not available on this operating system;
- '-': No data, incompatible collector version or the data is not yet available.

antivirus_up_to_date

enum

Determines whether to audit each instance of a user logging on to or logging off from another computer in which this computer is used to validate the account.

audit_account_logon_events

enum

Determines whether to audit each event of account management on a computer.

audit_account_management

enum

Determines whether to audit the event of a user accessing an Active Directory object that has its own system access control list (SACL) specified.

audit_directory_service_access

enum

Determines whether to audit each instance of a user logging on to or logging off from a computer.

audit_logon_events

enum

audit_object_access

	Determines whether to audit the event of a user accessing an object, e.g. a file, folder, registry key, printer, and so forth - that has its own system access control list (SACL) specified. enum
audit_policy_change	Determines whether to audit every incident of a change to user rights assignment policies, audit policies, or trust policies. enum
audit_privilege_use	Determines whether to audit each instance of a user exercising a user right. enum
audit_process_tracking	Determines whether to audit detailed tracking information for events such as program activation, process exit, handle duplication, and indirect object access. enum
audit_system_events	Determines whether to audit when a user restarts or shuts down the computer or when an event occurs that affects either the system security or the security log.
average_boot_duration	millisecond NU System boot duration baseline
average_logon_duration	millisecond NU User logon duration baseline
bios_serial_number	string NU BIOS serial number
chassis_serial_number	string NU Chassis serial number
collector_distinguished_name	string NU Indicates the distinguished name (DN) as seen: • For Windows: In Active Directory (AD). if no connection with AD is set up, a '-' is displayed;

	• For Mobile: In the Exchange ActiveSync server Note that this DN is reported by the Collector.
collector_installation_log	string NU Link to the last Nexthink Collector installation error log
collector_package_target_version	version NU Indicates the Collector package version that is targeted.
collector_status	enum NU Indicates the status of the Nexthink Collector package installed on the device: • unmanaged: the Collector is not automatically updated • up-to-date: the Collector is up-to-date • outdated: a newer Collector version is available.
collector_tag	integer Collector installation tag
collector_update_status	enum Current status of Nexthink Collector Updater
collector_version	version Version number of Nexthink Collector installation
cpu_frequency	mhz NU CPU frequency
cpu_model	string NU CPU model
database_usage	permill Percentage of the database used by information related with the device
device_encryption_required	boolean NU

	Indicates whether device encryption is required.
device_manufacturer	string NU Indicates the device manufacturer.
device_model	string NU Indicates the model of the device.
device_password_required	boolean NU Indicates whether a password is required on the device.
device_product_id	string NU Device product ID
device_product_version	string NU Device product version
device_serial_number	string NU Indicates the device serial number.
device_type	enum Type of device (desktop, laptop, server, mobile)
device_uid	md5 Indicates the universally unique identifier (based on Engine name and device ID)
device_uuid	string Indicates the device universally unique identifier (UUID)
directory_service_site	string NU Site (or location) of an Active Directory (AD) service
disks_manufacturers	string Hard disks manufacturers
disks_smart_index	percent NU Lowest S.M.A.R.T. index of installed hard disks (index is based on S.M.A.R.T. attributes)
distinguished_name	string NU Indicates the distinguished name (DN) as seen:

	• For Windows: In Active Directory (AD). if no connection with AD is set up, a '-' is displayed; • For Mobile: In the Exchange ActiveSync server
eas_access_state	enum Indicates whether the device can access the Exchange ActiveSync server. The possible states are: • allowed: the device has access; • blocked: the device is blocked; • discovery: the device is temporary quarantined while it is being identified by the Exchange ActiveSync server; • quarantined: the device is waiting for Exchange ActiveSync administrator approval.
eas_access_state_reason	enum Indicates the reason for the device access state. The possible values are: • global: caused by the global access settings; • device rule: caused by a device access rule; • individual: caused by an individual exemption; • policy: caused by Exchange ActiveSync policy.
eas_device_access_rule	string Indicates the name of the access rule. An access rule allows, blocks or quarantines devices based on the device

type, model, OS or user agent characteristics.

string

eas_device_identity

Indicates the identity of the device in Exchange ActiveSync Server.

enum

eas_exemption

Indicates whether a personal exemption is set for the device and its user. Possible values are:

- none;
- allow;
- block.

enum

eas_policy_application_status

Indicates whether the Exchange ActiveSync policy is applied or not. Possible values are:

- not applied;
- applied in full: the policy is applied (unless the field 'Allow non provisionable devices' value is 'yes');
- partially applied.

string

eas_policy_name

Indicates the name of the Exchange ActiveSync policy applied to the user's mailbox.

datetime

eas_policy_update

Indicates the last time the Exchange ActiveSync policy was updated on the device.

boolean NU

email_attachment_enabled

Indicates whether attachments can be downloaded to the mobile device through the Exchange ActiveSync protocol.

integer NU

enforce_password_history

Indicates the number of unique passwords that have to be associated with a user account before an old password can be reused.

entity	string Entity
extended_logon_duration_baseline	millisecond NU Extended logon duration baseline
firewall_name	string NU Name of the main firewall
firewall_rtp	enum Indicates whether the firewall real time protection (RTP) is active: • on: Indicates that RTP is active; • off: Indicates that either RTP is not active or no antivirus has been detected; • unknown: Indicates that the information could not be retrieved; • N/A: This field is not available on this operating system; • '-': No data, incompatible collector version or the data is not yet available.
first_seen	datetime NU Indicates the first time when the activity of the device was recorded: • For Windows and Mac OS: The first time Collector reported activity; • For Mobile: The first time the device was reported with a successful synchronization.
graphical_card_ram	byte NU Amount of RAM of the graphical card with most RAM
graphical_cards	string

	Installed graphical cards
group_name	string NU Name of computer domain or workgroup
guest_account_status	enum Determines if the Guest account is enabled or disabled.
hard_disks	string NC List of all hard disks
id	identifier Unique device identifier
internet_security_settings	enum Internet security settings (ok, at risk or unknown)
ip_addresses	ip_address List of IP addresses for the device
is_collector_distinguished_name_truncated	boolean Flag indicating whether the collector DN is truncated or not
is_directory_service_site_truncated	boolean Flag indicating whether the DS site is truncated or not
last_boot_duration	millisecond NU Duration of last system boot
last_extended_logon_duration	millisecond NU Last extended logon duration
last_ip_address	ip_address NU Last IP address assigned to the device
last_known_connection_status	enum NU Indicates the last known connection status of the device: • 'UDP': the device successfully connected via UDP but not TCP. • 'TCP': the device successfully connected via TCP but not UDP.

	• 'UDP+TCP': the device successfully connected via both UDP and TCP. • '-': Collector version is below V6.6.
last_logged_on_user	string NU Last logged on user
last_logon_duration	millisecond NU Last user logon duration
last_logon_time	datetime NU Last logon time
last_seen	datetime NU Indicates the last time that activity on the device was reported: • For Windows and Mac OS: The last time Collector reported activity through the UDP channel, • For Mobile: The last time the device successfully synchronized with the Mobile Bridge.
last_seen_on_tcp	datetime NU Indicates the last time that the device was successfully connected through the TCP channel. • '-': The Collector is an older version that does not support TCP.
last_system_boot	datetime NU Time of last system boot
last_update	datetime NU Indicates the last Collector update time.
last_update_status	enum NU Indicates the status of the last Collector update:

	• '-': the Collector was never updated • successful installation: the last Collector installation was successful • package download error: the Collector was not able to download the Collector package from Nexthink Appliance • package digital signature error: the Collector was not able to check the Collector package digital signature • device reboot required: the device needs to be rebooted to complete the Collector installation • package error: the Collector package installation has failed • internal error: the Collector package installation has failed for an unexpected reason.
last_updater_request	datetime NU Last time Nexthink Updater checked for updates
last_windows_update	datetime NU Time of last system Update
local_administrators	string Users and groups which are members of the Local Administrators group on the device.
local_power_users	string Users and groups which are members of the Local Powers Users group on the device.
logical_cpu_number	integer NU Indicates the number of cores multiplied by the number of threads that can run on

	each core through the use of hyperthreading.
logical_drives	string List of all logical drives
mac_addresses	mac_address List of MAC addresses for the device
maximum_password_age	integer NU Indicates the period in time (in days) during which the password can be used before the system requires the user to change it: • Windows: As set up in the group policy; • Mobile: As set up in security policies.
membership_type	enum Type of computer membership (domain/workgroup)
minimum_password_age	integer NU Period of time (in days) that a password must be used before the user can change it.
minimum_password_length	integer NU Least number of characters that a password for a user account may contain.
monitor_models	string Models of connected monitors
monitor_resolutions	string Screen resolutions of connected monitors
monitors	string Connected monitors
monitors_serial_numbers	string Serial numbers of connected monitors (ordered as in 'Monitors')
name	string Indicates the name of the device:

- For Windows: NetBios Name;
- For Mac OS: Computer name used on the network;
- For Mobile: Composed by mailbox name and device friendly name.

enum

Number of antispyware detected:

number_of_antispyware

- unknown: Indicates that the information could not be retrieved;
- N/A: This field is not available on this operating system;
- '-': No data, incompatible collector version or the data is not yet available.

enum

Number of antiviruses detected:

number_of_antiviruses

- unknown: Indicates that the information could not be retrieved;
- N/A: This field is not available on this operating system;
- '-': No data, incompatible collector version or the data is not yet available.

number_of_cores

integer NU

Number of cores

number_of_cpus

integer NU

Number of CPUs

number_of_days_since_first_seen

integer NU

Number of days since activity of the device was first recorded in the system.

number_of_days_since_last_boot

integer NU

Number of days since last system boot

number_of_days_since_last_eas_policy_update	integer NU Indicates the number of days since the last Exchange ActiveSync policy update.
number_of_days_since_last_logon	integer NU Number of days since last logon
number_of_days_since_last_seen	integer NU Indicates the number of days since the last time the device was seen by Nexthink. The field is updated whenever device activity is detected: • For Windows and Mac OS: seen through the UDP channel, • For Mobile: seen through the Mobile Bridge.
number_of_days_since_last_seen_on_tcp	integer NU Indicates the number of days since the last time the device was successfully connected through the TCP channel. '-': The Collector is an older version that does not support TCP.
number_of_days_since_last_windows_update	integer NU Number of days since last system Update
number_of_firewalls	enum Number of firewalls detected: • unknown: Indicates that the information could not be retrieved; • N/A: This field is not available on this operating system; • '-': No data, incompatible collector version or the data is not yet available.
number_of_graphical_cards	integer Number of installed graphical cards
number_of_monitors	integer

Number of connected monitors

enum

os_architecture

Architecture of device operating system (x86/x64)

version

os_build

Indicates the build number of the operating system.

string NU

Indicates name, version and architecture (when applicable) of the operating system.

os_version_and_architecture

- unknown: the OS version could not be retrieved or it could not be mapped to a recognized value.

enum

Indicates whether password complexity is required:

password_complexity_requirements

- Windows: The password must meet complexity requirements as defined in the group policy;
- Mobile: No simple passwords are allowed or a minimum password length is set, as defined in the security policy.

enum

Indicates the platform of the device. A platform is a set of operating system families on which the same objects, activities, events and properties can be retrieved. The possible values are:

platform

- Windows;
- Mac OS;
- Mobile.

privileges_of_last_logged_on_users enum

	Privileges of the last logged on user (user, power user, administrator)
sd_card_encryption_required	boolean NU Indicates whether SD card encryption is required.
sid	sid NU Windows security identifier for the device.
storage_policy	enum Indicates the event storage policy for the device. Possible values are: • all: web requests, connections and executions are stored • connections and executions; • executions; • none: no activity is recorded; • remove: The device will be removed from Engine during the next cleanup, as long as it is no longer sending data; Note that available events depend on the device platform.
system_drive_capacity	byte Total capacity of system drive
system_drive_free_space	byte Total available free space on system drive
system_drive_usage	percent NU Use percentage of system drive
total_active_days	day Total number of days the device was active.
total_drive_capacity	byte Total capacity of all drives

total_drive_free_space	byte Total free space on all drives
total_drive_usage	permill NU Total use percentage of all drives
total_nonsystem_drive_capacity	byte Total capacity of all non-system drives
total_nonsystem_drive_free_space	byte Total free space on all non-system drives
total_nonsystem_drive_usage	percent NU Total use percentage of all non-system drives
total_ram	byte NU Total amount of RAM
updater_error	string Last Nexthink Collector Updater error
updater_version	version Nexthink Collector Updater version
upgrade_group	enum NU Indicates the update group of Nexthink Collector: • manual: the Collector is manually updated • pilot: the Collector is updated as part of the pilot group • main: the Collector is updated as part of the main group.
user_account_control_status	enum User account control status (ok, at risk or unknown)
windows_license_key	string NU Windows license key
windows_updates_status	enum Windows update status (ok, at risk or unknown)

wmi_status	enum Windows WMI service status (ok, failure)

domain

A domain is a domain name e.g. www.nexthink.com. Platforms:

Name	Type	Properties
	permill	
database_usage	Percentage of the database used by information related with the domain	
	string	SE
domain_category	Indicates the category of the domain: • '-': Not yet tagged or internal domain.	
	datetime	NU
first_seen	The first time the domain has been seen.	
	string	SE
hosting_country	Indicates in which country the domain is hosted: • '-': Not yet tagged, internal domain or not known by Nexthink Library.	
	string	NU
hostname	The hostname of the fully qualified domain name	
id	identifier	
	Unique domain identifier	
internal_domain	boolean	
	Indicates whether the domain is considered internal: • yes: The domain is not reported to Nexthink Library and	

	subdomains are not compressed using the '*' pattern; • no: The domain is reported to the Nexthink Library (if the license includes the Security module); complex subdomains are compressed using the '*' pattern.
last_seen	datetime NU The last time the domain has been seen.
name	string The fully qualified domain name
protocol	enum Protocols used in web requests (HTTP, TLS, HTTP/TLS)
response_size	byte Total web incoming traffic
storage	enum Event storage policy for the domain (web request or none)
threat_level	enum SE Indicates the threat level of the domain: • '-': Not yet tagged or internal domain; • none detected: No known threat; • low: low threat; • intermediate: Intermediate threat; • high: High threat.

executable

An application is a executable programs e.g. 'winword.exe'. Platforms:

Name	Type	Properties
application_company	string	
	Application company	
application_name	string	
	Application name	
	permill	
database_usage	Percentage of the database used by information related with the executable.	
description	string	
	Executable description	
first_seen	datetime	NU
	First time activity of the executable was recorded on any device.	
id	identifier	
	Unique executable identifier	
known_packages	string	
	List of packages known to contain the executable. This list is not exhaustive: The presence of a package does not necessarily imply that on a given device the executable was installed through that package.	
last_seen	datetime	NU
	Last time activity of the executable was recorded on any device.	
name	string	
	Executable name	
platform	enum	
	The platform (operating system family) on which the executable is running.	
storage_policy	enum	

	Indicates the event storage policy for the executable. Possible values are: • all: web requests, connections and executions are stored; • connections and executions; • executions; • none: no activity is recorded.
total_active_days	day Total number of days the executable was active.

package

A package is a software packages (programs or updates). Platforms:

Name	Type	Properties
first_installation	datetime Time of first installation	NU
first_seen	datetime The first time the package has been seen.	NU
id	identifier Unique package identifier	
name	string Package name	
number_of_updates	integer Number of updates (for programs)	
platform	enum The platform (operating system family) on which the package is installed.	
program	string Package program	
publisher	string Package publisher	NU

status	enum Package status (installed/removed)
type	enum Package type (program/update)
version	string NU Package version
windows_7_32bit_compatibility	string Indicates the Windows 7 (32-bit) compatibility of the package: • '-': Not yet tagged; • No information available: Not known by Nexthink Library; • Compatible: Compatible with Windows 7.
windows_7_64bit_compatibility	string Indicates the Windows 7 (64-bit) compatibility of the package: • '-': Not yet tagged; • No information available: Not known by Nexthink Library; • Compatible: Compatible with Windows 7.

port

A port is a TCP or UDP connection ports. Platforms:

Name	Type	Properties
first_seen	datetime NU First time activity of the port was recorded on any device.	
id	identifier Unique port identifier	
last_seen	datetime NU	

	Last time activity of the port was recorded on any device.	
port_number	integer	
	Port number	
port_type	enum	
	Port type (tcp, udp, tcp port scan, udp port scan)	
port_value	port	
	Port value for tagging	

printer

A printer is an installed printers (local, network, shared or virtual). Platforms:

Name	Type	Properties
first_seen	datetime	NU
	First time activity of the printer was recorded on any device.	
host_name	string	
	Host name	
id	identifier	
	Unique print identifier	
last_seen	datetime	NU
	Last time activity of the printer was recorded on any device.	
location	string	NU
	Printer location	
model	string	
	Printer model	
name	string	
	Printer name	
real_name	string	
	Most frequently seen display name	
type	enum	
	Printer type (local/remote)	

service

A service represents an IT service in your organization, such as the mail service or the directory service. Services are either based on TCP connections (for Windows and Mac devices) or on web requests (for Windows devices only). Platforms:

Name	Type	Properties
id	integer Unique service identifier	
name	string Service name	
status	enum Service status (active, error)	
type	enum Type of service (network, web)	

url_path

A url_path is a URL path after the domain name e.g. [www.nexthink.com]/awards/. Platforms:

Name	Type	Properties
id	identifier Unique url path identifier	
path	string The URL path	

user

A user is an object that represents an individual account in a device (local user) or in a group of devices (domain user). The account may identify a physical user or a system user. Platforms:

Name	Type	Properties
database_usage	permill Percentage of the database used by information related with the binary	
department	string	

	User department as listed in active directory
distinguished_name	string NU Active directory distinguished name (DN)
first_seen	datetime NU First time activity of the user was recorded on any device.
full_name	string NU Full user name as listed in active directory
id	identifier Unique user identifier
job_title	string NU Job title as listed in active directory
last_seen	datetime NU Last time activity of the user was recorded on any device.
name	string User logon name
number_of_days_since_last_seen	integer NU Indicates the number of days since the last time the user was seen by Nexthink. The field is updated whenever user activity is detected.
seen_on_mac_os	boolean Indicates if the user has been seen on a Mac device.
seen_on_mobile	boolean Indicates if the user has been seen on a Mobile device.
seen_on_windows	boolean Indicates if the user has been seen on a Windows device.
sid	sid NU Indicates the Windows security identifier for the user. For Mac OS, '-' means that the user is not in Active Directory.

total_active_days	day Total number of days the user was active.
type	enum Type of user (local/domain/system)

Events

connection

A connection is a TCP connection or a UDP packet. Several identical TCP connections or UDP packets are merged when in close succession.

Platforms:

Name	Type	Properties
cardinality	integer Number of underlying connections, consolidated over time	
destination_ip_address	ip_address IP address of the connection destination	
device_ip_address	ip_address IP address of the connection source	
duration	millisecond The time between the start of the first connection and the end of the last underlying connection.	
end_time	datetime Connection end time, corresponding to the moment when the last underlying connection was closed.	
id	identifier Unique connection identifier	
incoming_bitrate	bps Average incoming bitrate of all underlying connections, consolidated over time	NU
incoming_traffic	byte Incoming traffic	

network_interface_iana_code	string (beta) Indicates the network interface IANA code.
network_interface_index	integer (beta) Indicates the network interface index.
network_interface_type	enum (beta) Indicates the network interface type. Possible values are: • wifi • ethernet • mobile • other • unknown: the Collector is not supporting interface type.
network_response_time	microsecond TCP connection establishment time
outgoing_bitrate	bps NU Average outgoing bitrate of all underlying connections, consolidated over time
outgoing_traffic	byte Outgoing traffic
start_time	datetime Connection start time
status	enum Status of the connection (established, rejected, no service, no host, closed)
type	enum Type of the connection (tcp, udp)

device_activity

A device_activity is a device activity (boot or activity).

Platforms:

Name	Type				Properties

	millisecond	
duration	Boot duration (timed between kernel start and launch of 'logonui.exe' process) or online duration	
id	identifier	
	Boot event identifier	
time	datetime	
	Time of boot	
type	enum	
	Activity event information	

device_error

A device_error is a critical system errors (system crash, hard reset, or disk error).

Platforms:

Name	Type	Properties
error_code	integer	
	Error code	
error_label	string	
	Error label	
id	identifier	
	Problem identifier	
start_time	datetime	
	Time of error	
type	enum	
	Indicates the device error type, with the following possible values: • system crash: a windows bluescreen of death; • hard reset: the device was abruptly stopped and then rebooted. It might be caused by pressing the reset button, a power failure or a crash;	

- SMART disk failure: a disk error was detected on a disk with SMART technology.

device_performance (Public Beta)

An device_performance reports the average IOPS, CPU and memory of a device during one hours.

Platforms:

Name	Type	Properties
average_cpu_usage	percent	
	Average CPU usage on the period	
average_memory_usage	byte	
	Average memory usage on the period	
duration	millisecond	
	Total report duration	
end_time	datetime	
	Report end time	
id	identifier	
	Unique report identifier	
read_bytes	byte	NU
	Total disk read bytes accumulated during the period	
read_operations	integer	NU
	Total disk read operations accumulated during the period	
time	datetime	
	Start time	
write_bytes	byte	NU
	Total disk write bytes accumulated during the period	
write_operations	integer	NU
	Total disk write operations accumulated during the period	

device_warning

A device_warning is a peak in device resource usage (CPU, memory or I/O).

Platforms:

Name	Type	Properties
duration	millisecond	
	Performance event duration	
end_time	datetime	
	Performance event end time	
id	identifier	
	Unique performance event identifier	
info	string	
	Performance event information	
start_time	datetime	
	Performance event start time	
type	enum Type of the device warning (high cpu usage, high io usage, high memory usage or high number of page faults).	
value	percent	
	Performance percentage	
warning_duration	millisecond Indicates the duration of the warning. This duration can be shorter than the event duration when the warning is not continuous.	

execution

An execution is a process executing on a device. Serveral executions of the same process are merged when in close succession.

Platforms:

Name	Type	Properties
average_memory_usage	byte	
	Average memory usage	

binary_path	path Executed binary path
cardinality	integer Number of underlying processes, consolidated over time
duration	millisecond Total execution duration
end_time	datetime Execution end time
id	identifier Unique execution identifier
incoming_tcp_traffic	byte Incoming TCP traffic
incoming_udp_traffic	byte Incoming UDP traffic
outgoing_tcp_traffic	byte Outgoing TCP traffic
outgoing_udp_traffic	byte Outgoing UDP traffic
privilege_level	enum Privilege level of the execution (user, power user, administrator)
start_time	datetime Execution start time
status	enum Status of the execution (started, stopped)
total_cpu_time	millisecond Total CPU time

execution_error

An execution_error is application errors (crash or not responding)

Platforms:

Name	Type				Properties

id	identifier Error identifier
info	string Error event information
time	datetime Time of error
type	enum Type of the execution error (application not responding, crash)

execution_warning

An execution_warning is a peak in application resource usage (CPU or memory).

Platforms:

Name	Type	Properties
duration	millisecond Performance event duration	
end_time	datetime Performance event end time	
id	identifier Unique performance event identifier	
info	string Performance event information	
start_time	datetime Performance event start time	
type	enum Type of the execution warning (high cpu usage, high memory usage)	
value	percent Performance percentage	
warning_duration	millisecond Indicates the duration of the warning. This duration can be shorter than the event duration when the warning is not continuous.	

installation

A installation is the installation or uninstallation of a Software packages (programs or updates).

Platforms:

Name	Type	Properties
id	identifier	
	Unique deployment identifier	
time	datetime	
	Installation start time	
type	enum	
	Type of operation (installation, uninstallation)	

network_scan

A network scan is a sequence of failed TCP connections or UDP packets made to the same port to more than 50 destinations within a few seconds.

Platforms:

Name	Type	Properties
cardinality	integer	
	Number of underlying connections, consolidated over time	
device_ip_address	ip_address	
	IP address of the connection source	
duration	millisecond	
	The time between the start of the first connection and end of the last underlying connection	
end_time	datetime	
	Scanning end time, corresponding to the moment when the last underlying connection was closed.	
id	identifier	
	Unique scanning identifier	
network	ip_network	

	Minimum IP network including all scanned destinations
start_time	datetime Scanning start time
status	enum Status of the Scanning (established, closed)
type	enum Type of the port scanning (tcp, udp)

port_scan

A port scan is a sequence of failed TCP connections or UDP packets made to the same destination to more than 50 ports within a few seconds.

Platforms:

Name	Type	Properties
cardinality	integer Number of underlying connections, consolidated over time	
destination_ip_address	ip_address IP address of the scanned destination	
device_ip_address	ip_address IP address of the connection source	
duration	millisecond The time between the start of the first connection and end of the last underlying connection.	
end_time	datetime Scanning end time, corresponding to the moment when the last underlying connection was closed.	
first_scanned_port	port First port scanning	
id	identifier Unique scanning identifier	
last_scanned_port	port Last port scanning	

start_time	datetime Scanning start time
status	enum Status of the Scanning (established, closed)
type	enum Type of the port scanning (tcp, udp)

printout

A printout is a print job processed by a printer.

Platforms:

Name	Type	Properties
color_print	boolean Color print	
document_type	string Type of printed document	
duplex	boolean Indicates whether the pages are printed on both sides of the sheet.	
id	identifier Unique print job identifier	
number_of_printed_pages	integer Number of printed pages	NU
page_size	string Paper size for printed pages	
print_quality	enum Print quality	
size	byte Print job size in bytes	NU
status	enum Print job status(success, error, timeout)	
time	datetime Print job time	

user_activity

A user_activity is a user activity (logon or interactive activity).

Platforms:

Name	Type	Properties
	millisecond	
duration	Indicates the time between the user logging on and the desktop being shown.	
id	identifier	
	User logon event identifier	
	millisecond	
real_duration	Indicates the time between the user logging on and the device being ready to use. Desktops and laptops are considered fully functional once the CPU usage drops below 15% and the disk usage drops below 80%, and servers once the CPU usage of all processes belonging to the corresponding user drops below 15%.	
time	datetime	
	Time of user logon	
type	enum	
	Activity event information	

web_request

A web_request is a HTTP or TLS requests.

Platforms:

Name	Type	Properties
	integer	
cardinality	Number of underlying web requests, consolidated over time	
	millisecond	
connections_duration	The time between start of the first connection and end of the last underlying connection	

	datetime	
end_time	Web request end time, corresponding to the moment when the last underlying TCP connection was closed.	
http_status	http_status_code	NU
	HTTP response status code	
id	identifier	
	Unique request identifier	
incoming_traffic	byte	
	Incoming web traffic of all underlying web requests, consolidated over time	
	microsecond	
network_response_time	Average TCP connection establishment time of all underlying connections, consolidated over time	
outgoing_traffic	byte	
	Outgoing web traffic of all underlying web requests, consolidated over time	
protocol	enum	
	Web request protocol (HTTP, TLS)	
protocol_version	enum	
	Web request protocol version	
	boolean	
service_related	Indicates whether the web request is related to a configured service: • yes: These requests are always visible by all users; • no: Depending on the privacy settings, requests not related to a service might not be visible by everyone.	
start_time	datetime	
	Web request start time	
	millisecond	
web_request_duration	Average time between request and last response byte of all underlying requests, consolidated over time	

Relationships

A relationships is a link between object and event tables and is specified in a **with** clause.

connection

- device
- user
- binary
- executable
- application
- destination
- port
- service

device_activity

- device

device_error

- device

device_performance

- device
- user

device_warning

- device

execution

- device
- user
- binary
- executable
- application

execution_error

- device
- user
- binary
- executable
- application

execution_warning

- device
- user
- binary

- executable
- application

installation

- device
- package

network_scan

- device
- user
- binary
- executable
- application
- port

port_scan

- device
- user
- binary
- executable
- application
- destination

printout

- device
- user
- printer

user_activity

- device
- user

web_request

- device
- user
- binary
- executable
- application
- destination
- port
- domain
- url_path
- service

package

- device
- package

Aggregates

connection

Name	Type				Properties
number_of_devices	integer				FP
	Number of devices				
number_of_users	integer				FP
	Number of users				
number_of_applications	integer				FP
	Number of applications				
number_of_executables	integer				FP
	Number of executables				
number_of_binaries	integer				FP
	Number of binaries				
number_of_destinations	integer				
	Number of destinations				
number_of_ports	integer				
	Number of ports				
number_of_connections	integer				
	Number of connections				
cumulated_connection_duration	millisecond				
	Cumulated duration of TCP connections				
activity_start_time	datetime				NU
	Start time of investigated activity				
activity_stop_time	datetime				NU
	Stop time of investigated activity				
incoming_traffic	byte				NU
	Total network incoming traffic				
outgoing_traffic	byte				NU
	Total network outgoing traffic				

average_network_response_time	microsecond	
	Average TCP connection establishment time	
successful_connections_ratio	permill	NU
	Percentage of successful TCP connections	
network_availability_level	availability_level	NU
	Graded ratio of successful TCP connections (high, medium, low)	
average_incoming_bitrate	bps	NU
	Average incoming network bitrate	
average_outgoing_bitrate	bps	NU
	Average outgoing network bitrate	
highest_local_privilege_reached	privileges_level	NU
	Highest local privilege level reached for executions (user, power user, administrator)	
number_of_events	integer	NU
	Number of events	
incoming_network_traffic_per_device	byte	NU
	Device average incoming network traffic	
outgoing_network_traffic_per_device	byte	NU
	Device average outgoing network traffic	
total_network_traffic	byte	NU
	Network traffic	

device_activity

Name	Type	Properties
number_of_devices	integer	
	Number of devices	
average_boot_duration	millisecond	NU
	Average system boot duration	

average_logon_duration	millisecond NU Average user logon duration
average_extended_logon_duration	millisecond NU Average extended logon duration
number_of_boots	integer Number of system boots
activity_start_time	datetime NU Start time of investigated activity
activity_stop_time	datetime NU Stop time of investigated activity
uptime	millisecond NU Amount of time the machine has been running
cumulated_interaction_duration	millisecond NU Cumulated time with user interaction (mouse or keyboard events)
number_of_events	integer NU Number of events

device_error

Name	Type	Properties
number_of_devices	integer Number of devices	
number_of_errors	integer Number of system errors	
activity_start_time	datetime NU Start time of investigated activity	
activity_stop_time	datetime NU Stop time of investigated activity	
number_of_events	integer NU Number of events	

device_performance

Name	Type	Properties
total_read_bytes	byte	NU/PB
	Total read bytes	
total_write_bytes	byte	NU/PB
	Total write bytes	
total_read_operations	integer	NU/PB
	Average read IPOS	
total_write_operations	integer	NU/PB
	Average write IPOS	
cumulated_measured_duration	millisecond	NU/PB
	Average read/write IPOS	
average_memory_usage	byte	NU/PB
	Average memory usage	
average_cpu_usage	percent	NU/PB
	Average CPU usage	
number_of_events	integer	NU/PB
	Number of events	

device_warning

Name	Type	Properties
number_of_devices	integer	
	Number of devices	
number_of_warnings	integer	
	Number of warnings	
cumulated_warning_duration	millisecond	NU
	Cumulated duration of the warning events	
activity_start_time	datetime	NU
	Start time of investigated activity	
activity_stop_time	datetime	NU
	Stop time of investigated activity	
number_of_events	integer	NU

	Number of events permill NU
high_device_overall_cpu_time_ratio	Indicates the ratio between the time the device is in high overall CPU usage and its uptime. permill NU
high_device_memory_time_ratio	Indicates the ratio between the time the device is in high memory usage and its uptime. permill NU
high_device_io_throughput_time_ratio	Indicates the ratio between the time the device is in high IO throughput and its uptime. permill NU
high_device_page_faults_time_ratio	Indicates the ratio between the time the device is in high page faults and its uptime.

execution

Name	Type	Properties
number_of_devices	integer Number of devices	FP
number_of_users	integer Number of users	FP
number_of_applications	integer Number of applications	FP
number_of_executables	integer Number of executables	FP
number_of_binaries	integer Number of binaries	FP
number_of_executions	integer Number of executions	

cumulated_execution_duration	millisecond	✓	✓		NU
	Cumulated duration of executions				
activity_start_time	datetime	✓	✓		NU
	Start time of investigated activity				
activity_stop_time	datetime	✓	✓		NU
	Stop time of investigated activity				
incoming_traffic	byte	✓	✓		NU
	Total network incoming traffic				
outgoing_traffic	byte	✓	✓		NU
	Total network outgoing traffic				
highest_local_privilege_reached	privileges_level Highest local privilege level reached for executions (user, power user, administrator)	✓	✓		NU
number_of_events	integer	✓	✓		NU
	Number of events				
average_memory_usage_per_execution	byte	✓	✓		NU
	Average memory usage per execution				
cpu_usage_ratio	permill	✓	✓		NU
	Average CPU usage				
total_cpu_time	millisecond	✓	✓		NU
	Total CPU time				
incoming_network_traffic_per_device	byte	✓	✓		NU
	Device average incoming network traffic				
outgoing_network_traffic_per_device	byte	✓	✓		NU
	Device average outgoing network traffic				
total_network_traffic	byte	✓	✓		NU
	Network traffic				

execution_error

Name	Type				Properties
application_not_responding_event_ratio	permill	✓	✓		NU
	Application not responding event ratio				

application_crash_ratio	permill NU Application crash ratio
number_of_application_not_responding_events	integer Number of application not responding events
number_of_application_crashes	integer Number of application crashes
number_of_devices	integer Number of devices
number_of_users	integer Number of users
number_of_applications	integer Number of applications
number_of_executables	integer Number of executables
number_of_binaries	integer Number of binaries
number_of_errors	integer Number of errors
activity_start_time	datetime NU Start time of investigated activity
activity_stop_time	datetime NU Stop time of investigated activity
number_of_events	integer NU Number of events

execution_warning

Name	Type	Properties
number_of_devices	integer Number of devices	
number_of_users	integer Number of users	
number_of_applications	integer	

	Number of applications
number_of_executables	integer
	Number of executables
number_of_binaries	integer
	Number of binaries
number_of_warnings	integer
	Number of warnings
cumulated_warning_duration	millisecond NU
	Cumulated duration of the warning events
activity_start_time	datetime NU
	Start time of investigated activity
activity_stop_time	datetime NU
	Stop time of investigated activity
number_of_events	integer NU
	Number of events
high_application_thread_cpu_time_ratio	permill NU
	High application thread CPU time ratio

installation

Name	Type	Properties
number_of_packages	integer	
	Number of packages	
number_of_devices	integer	
	Number of devices	
activity_start_time	datetime NU	
	Start time of investigated activity	
activity_stop_time	datetime NU	
	Stop time of investigated activity	
number_of_installations	integer	
	Number of installations	
number_of_events	integer NU	
	Number of events	

network_scan

Name	Type	Properties
number_of_devices	integer	
	Number of devices	
number_of_users	integer	
	Number of users	
number_of_applications	integer	
	Number of applications	
number_of_executables	integer	
	Number of executables	
number_of_binaries	integer	
	Number of binaries	
number_of_ports	integer	
	Number of ports	
number_of_connections	integer	
	Number of connections	
cumulated_scan_duration	millisecond	NU
	Cumulated duration of the network scan	
activity_start_time	datetime	NU
	Start time of investigated activity	
activity_stop_time	datetime	NU
	Stop time of investigated activity	
incoming_traffic	byte	NU
	Total network incoming traffic	
outgoing_traffic	byte	NU
	Total network outgoing traffic	
number_of_events	integer	NU
	Number of events	
incoming_network_traffic_per_device	byte	NU
	Device average incoming network traffic	
outgoing_network_traffic_per_device	byte	NU
	Device average outgoing network traffic	

total_network_traffic	byte Network traffic	■	●	▯	NU	

package

Name	Type	■	●	▯	Properties
number_of_devices	integer Number of devices	■	●	▯	FP
number_of_packages	integer Number of packages	■	●	▯	FP

port_scan

Name	Type	■	●	▯	Properties
number_of_devices	integer Number of devices	■	●	▯	
number_of_users	integer Number of users	■	●	▯	
number_of_applications	integer Number of applications	■	●	▯	
number_of_executables	integer Number of executables	■	●	▯	
number_of_binaries	integer Number of binaries	■	●	▯	
number_of_connections	integer Number of connections	■	●	▯	
number_of_destinations	integer Number of destinations	■	●	▯	
cumulated_scan_duration	millisecond Cumulated duration of the network scan	■	●	▯	NU
activity_start_time	datetime	■	●	▯	NU

activity_stop_time	Start time of investigated activity datetime NU Stop time of investigated activity
incoming_traffic	byte NU Total network incoming traffic
outgoing_traffic	byte NU Total network outgoing traffic
number_of_events	integer NU Number of events
incoming_network_traffic_per_device	byte NU Device average incoming network traffic
outgoing_network_traffic_per_device	byte NU Device average outgoing network traffic
total_network_traffic	byte NU Network traffic

printout

Name	Type	Properties
number_of_devices	integer Number of devices	
number_of_users	integer Number of users	
number_of_printers	integer Number of printers	
number_of_printed_pages	integer Number of printed pages	
number_of_printouts	integer Number of print jobs	
activity_start_time	datetime NU Start time of investigated activity	

activity_stop_time	datetime NU Stop time of investigated activity
number_of_events	integer NU Number of events

user_activity

Name	Type	Properties
number_of_devices	integer Number of devices	
number_of_users	integer Number of users	
number_of_logons	integer Number of user logons	
activity_start_time	datetime NU Start time of investigated activity	
activity_stop_time	datetime NU Stop time of investigated activity	
cumulated_interaction_duration	millisecond NU Cumulated time with user interaction (mouse or keyboard events)	
average_logon_duration	millisecond NU Average user logon duration	
average_extended_logon_duration	millisecond NU Average extended logon duration	
number_of_events	integer NU Number of events	

web_request

Name	Type	Properties
total_web_traffic	byte	NU
	Web traffic	
outgoing_web_traffic_per_device	byte	NU
	Outgoing web traffic per device	
incoming_web_traffic_per_device	byte	NU
	Incoming web traffic per device	
number_of_devices	integer	FP
	Number of devices	
number_of_domains	integer	FP
	Number of domains	
number_of_users	integer	FP
	Number of users	
number_of_applications	integer	FP/NU
	Number of applications	
number_of_executables	integer	FP
	Number of executables	
number_of_binaries	integer	FP
	Number of binaries	
number_of_destinations	integer	
	Number of destinations	
number_of_ports	integer	
	Number of ports	
activity_start_time	datetime	NU
	Start time of investigated activity	
activity_stop_time	datetime	NU
	Stop time of investigated activity	
average_network_response_time	microsecond	
	Average TCP connection establishment time	
highest_local_privilege_reached	privileges_level	NU
	Highest local privilege level reached for executions (user, power user, administrator)	
number_of_web_requests	integer	

	Number of web requests
protocols_used_in_requests	web_protocol_combination NU Protocols used in web requests (HTTP, TLS, HTTP/TLS)
lowest_protocol_version	min_web_protocol_version NU Lowest protocol version observed in web requests (excluding web requests with unknown protocol version)
incoming_traffic	byte NU Total web incoming traffic
outgoing_traffic	byte NU Total web outgoing traffic
average_incoming_bitrate	bps NU Average incoming bitrate of all underlying web requests, consolidated over time
average_outgoing_bitrate	bps NU Average outgoing bitrate of all underlying web requests, consolidated over time
cumulated_web_request_duration	millisecond NU Cumulated duration of web requests
cumulated_web_interaction_duration	millisecond NU Cumulated time during which web requests occurred, counted with a 5 minutes resolution.
average_request_size	byte NU Average size of web requests
average_response_size	byte NU Average size of web responses
average_request_duration	millisecond Average time between request and last response byte
successful_http_requests_ratio	permill NU Percentage of successful HTTP requests (1xx, 2xx and 3xx)
number_of_events	integer NU Number of events

Definitions

The following document lists all objects, fields and aggregates available through NXQL. Each field and aggregate have a name, a type, properties and a description.

Platforms can have the following values:

- **W**: The field, aggregate or table is available on the Windows platform.
- **X**: The field, aggregate or table is available on the Mac OS platform.
- **M**: The field, aggregate or table is available on the Mobile platform.

Properties can have the following values:

- **DE**: The field or aggregate is deprecated.
- **PB**: The field or aggregate is in Public Beta.
- **FP**: The field or aggregate can be used without a between clause.
- **NU**: The field or aggregate can be nil.
- **SE**: The field or aggregate is only available with a license containing the **security** feature.
- **WE**: The field or aggregate is only available with a license containing the **web monitoring** feature.
- **NC**: The field is not comparable.

Examples and tools

Excel integration with NXQL

This example shows how to query the Engine from Excel using NXQL. It replicates the functionality of the NXQL web editor included in every Engine that has the Integration tookit in an Excel spreadsheet. The provided macros run the queries that you type in and store their results in a separate sheet of your choice.

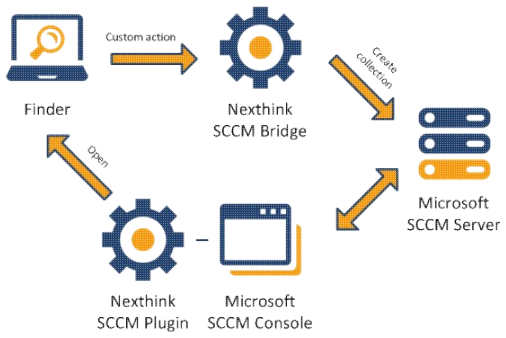
Explore the code and learn how to integrate NXQL calls into reports automatically generated with Excel.

Click to download the example of Excel integration with NXQL.

Integrating with SCCM

Overview

Export lists of devices or users from the results of investigations in Nexthink to new or existing collections in SCCM. From the System Center 2012 Configuration Manager Console, launch predefined investigations on users or devices in the Finder.



Download from here the installer for the Nexthink integration with SCCM. Note that you must have purchased the Integration toolkit module to download the installer.

Console extensions

Once you have installed the Nexthink integration with SCCM, find the Nexthink button at the **Home** tab of the SCCM console. When viewing users or devices in the SCCM console, press the Nexthink button and launch one of the predefined actions on the selected users or devices. The Finder executes an investigation or displays the device or the user view, depending on the chosen action.

The Nexthink button is also accessible from the context menu that pops up when you right-click a selection of users or devices.

The bridge

After executing an investigation in the Finder that returns a list of users or devices, select one or more of the returned items and right-click on them to bring up a context menu. In the context menu, select **Custom actions > Export to SCCM...** to export the selected items as a new or existing collection to SCCM.

Integrating with ServiceNow

CMDB Connector

Synchronize the Configuration Items that you see in Nexthink with the CMDB of your ServiceNow instance.

Find the application in the official ServiceNow Store, purchase it for free, and install it in your ServiceNow instance. Download from here the documentation and the associated content pack.

Incident Management Connector

Integrate end-user analytics from Nexthink into the incident management system of ServiceNow for improved Help Desk support.

Find the application in the official ServiceNow Store, purchase it for free, and install it in your ServiceNow instance. Download from here the documentation about the integration.

Integrating with HP ArcSight

The Nexthink integration with ArcSight lets you send global alerts triggered by conditions on device or binary objects to your ArcSight server via syslog messages. The ArcSight server receives these alerts as events in the Common Event Format (CEF), letting you compare and correlate Nexthink alerts with other types of CEF events sent by third-party products.

The Nexthink integration with ArcSight is a certified HP integration.

Download from here the documentation and software deliverables.