Purpose

This document describes how to use the Nagios XI SNMP Trap Interface (NXTI) to monitor and manage incoming SNMP Traps.

Target Audience

This document is intended for use by Nagios Administrators looking to manage their SNMP Trap definitions through the web interface.

Overview

NXTI was introduced with Nagios XI 5.5 and is enabled with the Enterprise edition license of Nagios XI. If you do not have the Enterprise edition license you can still configure Nagios XI to accept SNMP traps however this is not covered here, please refer to the Integrating SNMP Traps With Nagios XI documentation.

NXTI serves as a web front-end to the `snmptrapd`/`snmptt` workflow configuration. Previously this was configured by manually editing configuration files *(still the required method if you don't have the Enterprise edition license)*. NXTI additionally provides reporting of the received traps which are stored in a database, this functionality is exclusive to NXTI.

What Is An SNMP Trap?

Taken from the Net-SNMP website:

> A TRAP is a SNMP message sent from one application to another (which is typically on a remote host). Their purpose is merely to notify the other application that something has happened, has been noticed.

An important point to stress with SNMP traps is that they are asynchronous events that can occur at any time, in Nagios XI this is what is called a **PASSIVE** check/service. This means that they are not actively checked by Nagios XI on a regular schedule, Nagios XI is waiting for a SNMP Trap to be received from the remote device.

A comparison between an active check and a passive check helps explain the difference between **ACTIVE** and **PASSIVE** checks:

Scenario: UPS device loses input power and is running on batteries.

- With an **ACTIVE** check, if Nagios XI was checking the device on a 5 minute interval then it might be up to 5 minutes before Nagios XI is aware that the device is on batteries
- With a **PASSIVE** check, the device immediately sends an SNMP Trap to Nagios XI when it is running on batteries

More detailed information on passive service can be found in the following documentation:
Configuring Passive Services With Nagios XI

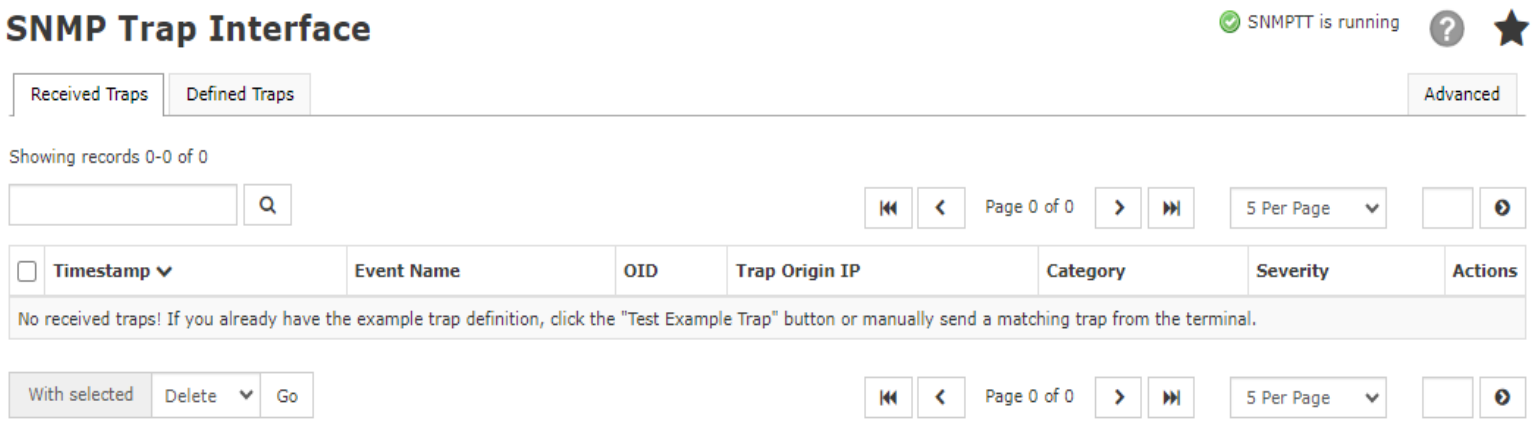
SNMP v2 vs SNMP v3

SNMP traps can be received using v2 or v3 of the protocol. By default the Nagios XI server will accept inbound SNMP v2 traps from any device. Security for accepting SNMP v2 traps is explained in the following KB article:
Nagios XI - SNMP Trap Hardening

Nagios XI needs to be configured before it can accept SNMP v3 traps, this is detailed in the following KB article:
Nagios XI - SNMP Trap v3 Configuration

NXTI Interface

To access NXTI navigate to **Admin** > **Monitoring Config** > **SNMP Trap Interface**.



NXTI provides the following capabilities:

- View / Add / Edit / Copy / Delete / Disable trap definitions
- View / Delete received trap logs
- Search and sort both trap definitions and received trap logs
- Monitor the `snmptt` process
- Locally test `snmptrapd`/`snmptt` functionality

The SNMP Trap interface is only accessible to Nagios admins, non-admin users will not have access.

How NXTI Works

NXTI utilizes the SNMPTT application that is provided with Nagios XI, this is how the operating system processes the received traps into useful data. If you have previously worked with the SNMPTT configuration files then you will be aware that it can get quite complex. NXTI provides a simple way to add /edit / remove trap definitions to the SNMPTT configuration.

The default SNMPTT configuration file on your Nagios XI server is `/etc/snmp/snmptt.conf` and is where the non-NXTI trap configurations reside. NXTI utilizes the separate configuration file `snmptt.conf.nxti`, this file should never be edited manually as those changes will be lost. Whenever you add / edit / remove a trap in NXTI `snmptt.conf.nxti`, is updated automatically and the `snmptt` service is restarted.

The trap definitions created by NXTI adhere to the SNMPTT configuration file format, hence this documentation will explain how the NXTI fields relate to the SNMPTT trap definitions in the configuration file.

Every trap that is defined in a SNMPTT configuration file begins with an `EVENT` line, this is how the incoming trap is matched with a trap definition. Here is an example:

```
EVENT NXTI_Event_1 NET-SNMP-EXAMPLES-MIB::netSnmpExampleHeartbeatNotification
"NXTI Test Event" Normal
```

If an incoming trap is matched against the OID in the `EVENT` line then the `EXEC` line(s) defined are executed (along with the other optional features of SNMPTT). The `EXEC` lines are how incoming trap data is actioned.

The most basic functionality that NXTI provides is to store a received trap in the database; these traps can be queried at any time after they have been received. While this basic configuration does not provide you with notifications for the received traps, it does allow you to receive a broad range of trap data that you can analyze without generating unnecessary notifications. This data is added to the database with the following EXEC command *(just a sample of the line is shown)*:

```
EXEC php /usr/local/nagiosxi/scripts/nxti.php --event_name="$N" ....
```

To receive notifications for received traps then you can use the **Passive Service Setup** component of the trap definition, this adds an EXEC command similar to *(just a sample of the line is shown)*:

```
EXEC /usr/local/bin/snmptraphandling.py "$aR" "SNMP Traps" ....
```

Furthermore you can define additional EXEC commands, this allows you to take other required actions for the received trap. Here is an example where a line is appended to a text file:

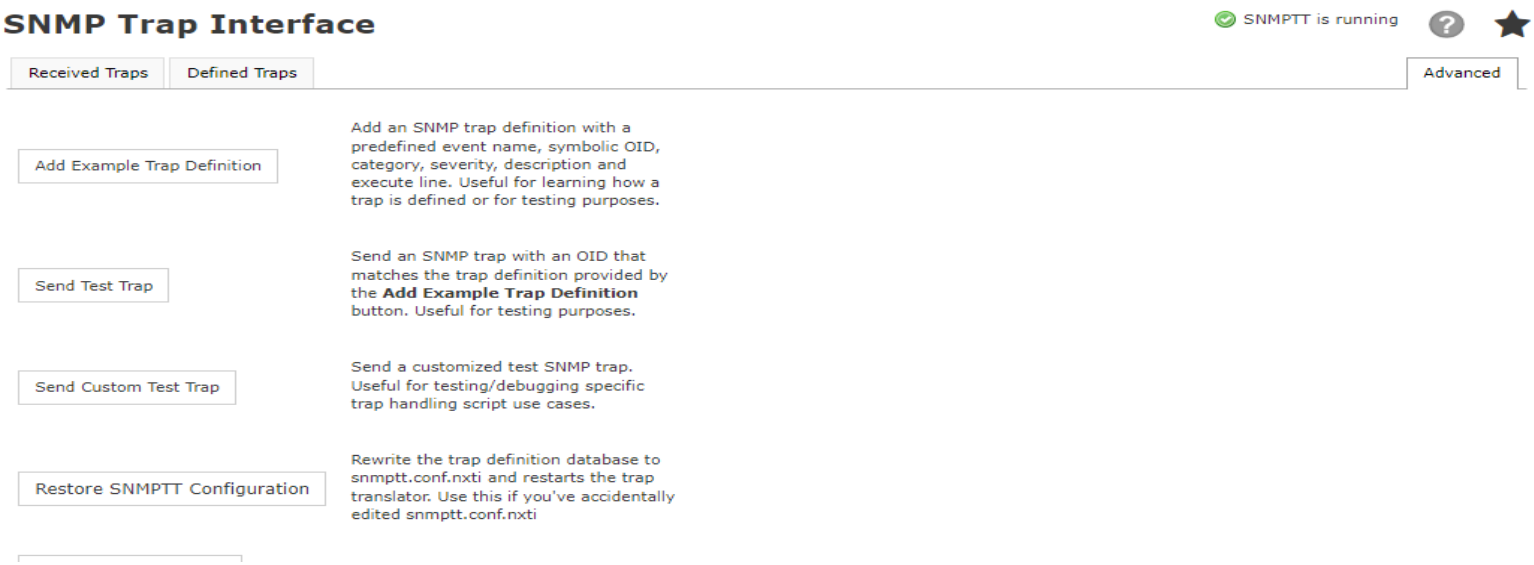
```
EXEC echo 'Success!' >> /usr/local/nagiosxi/var/NXTI_Write_Test
```

Finally, the advanced capabilities of SNMPTT can be defined such as `PREEXEC`, `NODES`, `MATCH`, `REGEX`. These are outside the scope of this documentation however the following guide does explain how MATCH can be utilized:

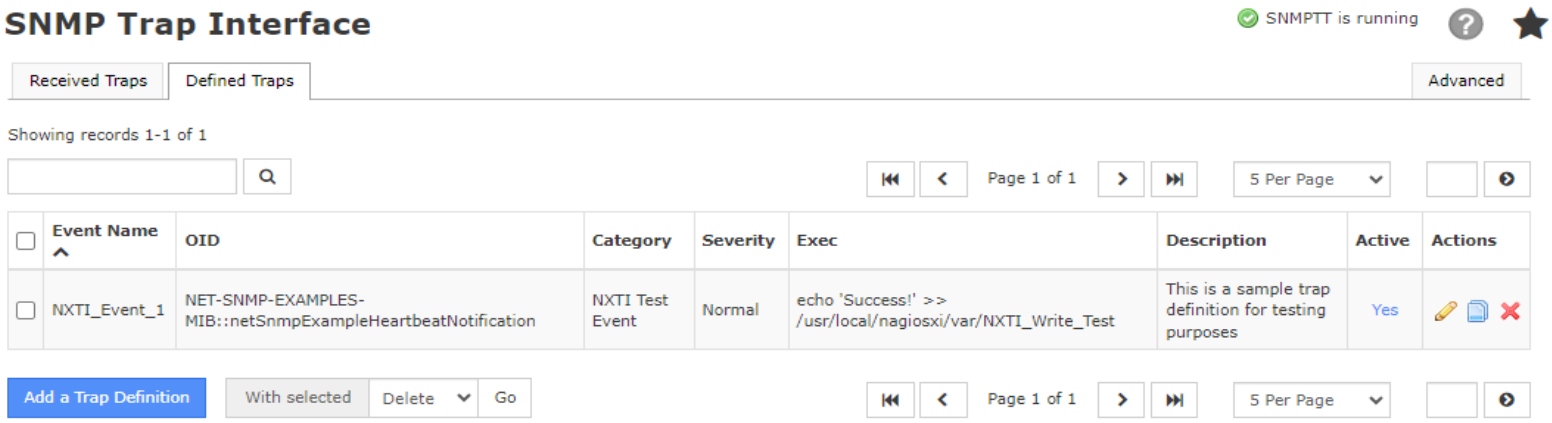
Nagios XI - SNMP Trap Tutorial

Adding Trap Definitions

The **Defined Traps** tab allows you to create a trap definition and has many options available. For a beginner these options can be overwhelming and an example really helps learn how it works. The **Advanced** tab provides an **Add Example Trap Definition** button that does just that.



Once you click the button a message will appear at the top of the screen telling you it was added. Click the **Defined Traps** tab to see the newly added trap definition.



The trap that was added demonstrates how you can append some text to a file on the Nagios XI server when a heartbeat trap is received. Click the **edit** icon in the **Actions** column to edit the trap. This opens the **Edit Trap Definition** page, which is almost identical to the **Add a Trap Definition** tab.

Here is that example trap definition as it exists in the `snmptt.conf.nxti` file.

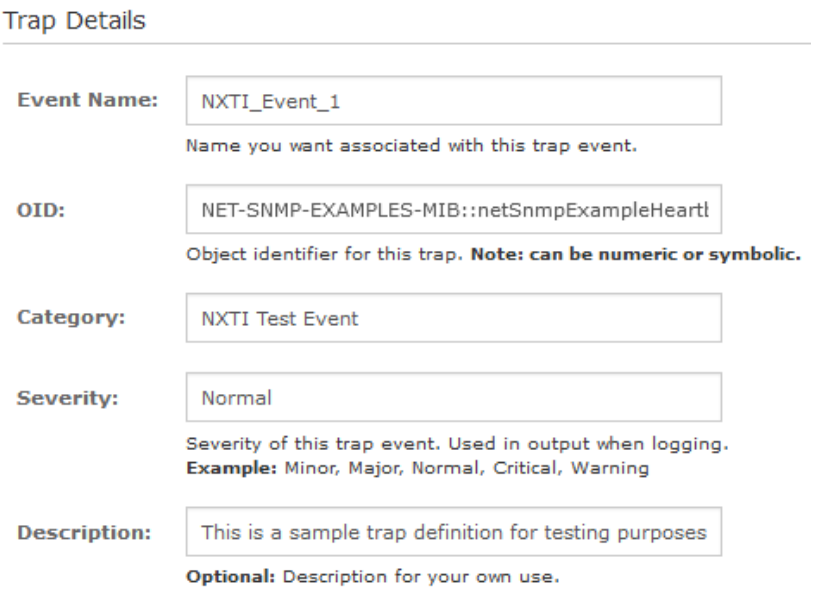
```
EVENT NXTI_Event_1 NET-SNMP-EXAMPLES-MIB::netSnmpExampleHeartbeatNotification
"NXTI Test Event" Normal
FORMAT Received trap "$N" with variables "$+*"
EXEC php /usr/local/nagiosxi/scripts/nxti.php --event_name="$N"
--event_oid="$i" --numeric_oid="$o" --symbolic_oid="$O" --community="$C"
--trap_hostname="$R" --trap_ip="$aR" --agent_hostname="$A" --agent_ip="$aA"
--category="$c" --severity="$s" --uptime="$T" --datetime="$x $X" --bindings="$
+*"
EXEC echo 'Success!' >> /usr/local/nagiosxi/var/NXTI_Write_Test
SDESC
This is a sample trap definition for testing purposes
EDESC
```

There are four components to adding a trap definition:

- Trap Details
  - This is how a trap is identified and classified
- Passive Service Setup
  - Configure the trap so that Nagios XI can receive the trap data in a service
- Exec
  - Define commands to be executed when the trap is received
- Advanced
  - For the advanced capabilities of SNMPTT such as `PREEXEC`, `NODES`, `MATCH`, `REGEX`

Trap Details

The fields **Event Name**, **OID**, **Category** and **Severity** are specifically for the EVENT line in a trap definition and are required.



These directives are mandatory:

- **Event Name**
  - This must be a unique name that cannot contain spaces
- **OID**
  - This is how an incoming trap is matched against this trap definition
  - Can be either a full numeric OID or a symbolic OID
  - Numeric OID is the raw (hard to read) format, for example:
    - `.1.3.6.1.4.1.8072.2.3.0.1`
  - Symbolic OID is an easy to read version of the numeric OID (case-sensitive)

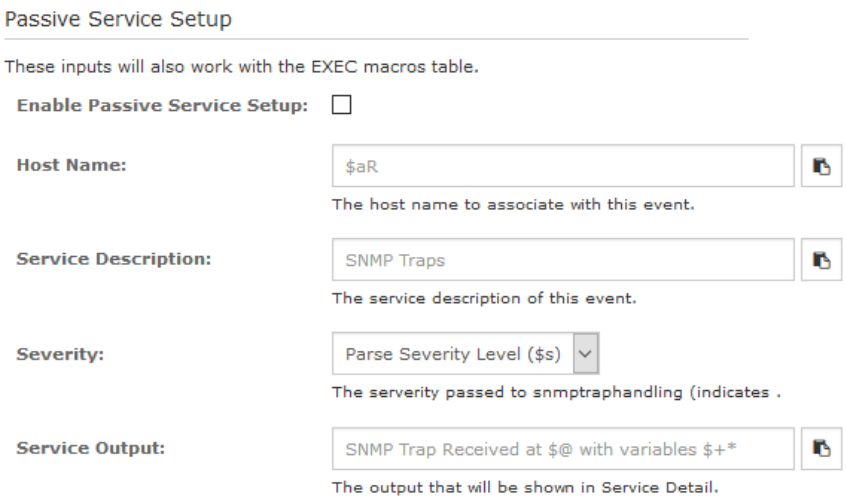
  - You can define the fully-qualified MIB name, for example:
    - `iso.org.dod.internet.private.enterprises.netSnmp.netSnmpExamples.netSnmpExampleNotifications.netSnmpExampleNotificationPrefix.netSnmpExampleHeartbeatNotification`
  - You can also use a shorter variant, for example:
    - `NET-SNMP-EXAMPLES-MIB::netSnmpExampleHeartbeatNotification`
- **Category**
  - Allows you to categorize the incoming trap as per your requirements
  - The options like `IGNORE` and `LOGONLY` should be avoided (see documentation)
- **Severity**
  - Typically has a value from one of: "Minor", "Major", "Normal", "Critical", "Warning"
  - Cannot contain spaces
  - This field value correlates to the Passive Service Setup severity option of **Pass Severity Level**

The **Description** field corresponds to the text between the `SDESC` and `EDESC` lines.

- This is used to describe the conditions and handling of the event to technicians
- This can contain any text
- Optional

\Passive Service Setup

This section is how you can configure this trap definition to send a passive check result to a Nagios service. The point of doing this is so that your Nagios XI users can receive notifications when traps are received for this trap.



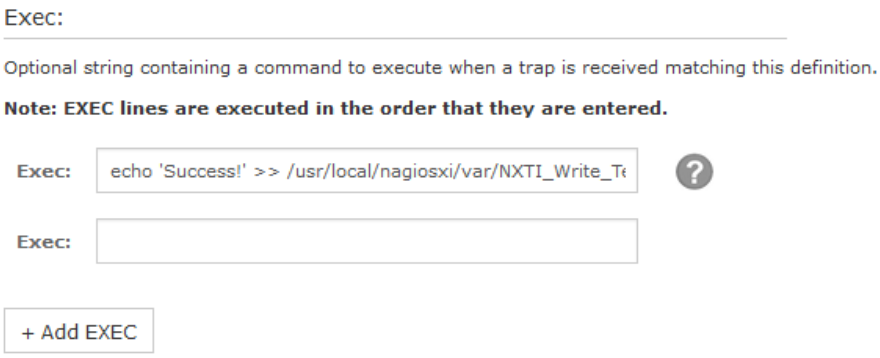
Check the box to enable this functionality, you will also be required to populate each field, the values already provided in each field are sufficient however you are required to click the placeholder icon to the right of the field to use it.

For the **Check Status** selection, this is what will be passed to Nagios as the state of the passive check result. The **Pass Severity Level** option is commonly used as it does not hard code the state into the definition. This allows for advanced configurations where you have multiple trap definitions with identical OIDs but are using `MATCH` options to differentiate them. An example of this can be found in the [SNMP Trap Tutorial](#) KB article.

When you enable Passive Service Setup, an `EXEC` line is added to `snmptt.conf.nxti` for this definition.

Exec

Each Exec entry corresponds to an `EXEC` line, these are the commands that get executed by the `snmptt` process when each event occurs.

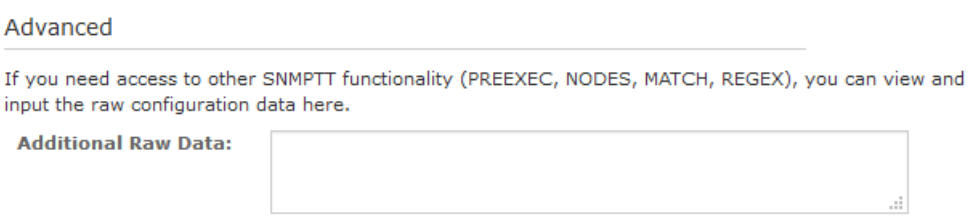


By clicking the question mark to the right of the first Exec entry, you can see the list of macros that can be used (these are built in to SNMPTT).

You can add as many `EXEC` lines as required, simply click the **+ Add EXEC** button to make another field appear. **Removing** an `EXEC` line is done by clearing the contents of the field.

Advanced

The **Advanced** section allows you to use other features of SNMPTT such as `PREEXEC`, `NODES`, `MATCH`, `REGEX`.

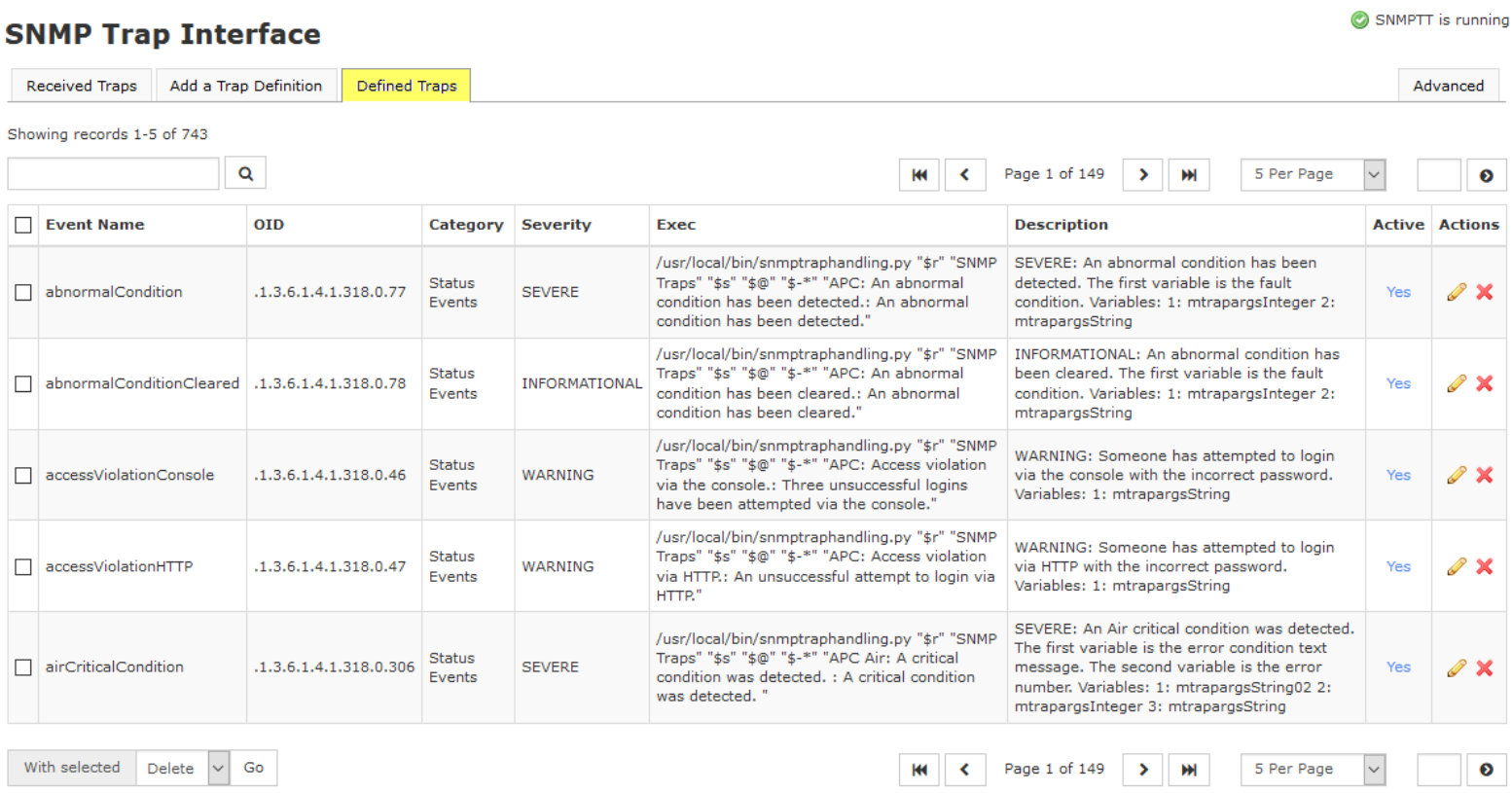


These are commonly used to manipulate the received trap data before the EXEC lines are executed. A detailed example using a `MATCH` can be found in the [SNMP Trap Tutorial](#) KB article.

Clicking the **Submit** button will save the changes to the `snmptt.conf.nxti` file and restart the `snmptt` service. Clicking the **Back** button will discard any changes you have made.

Managing Trap Definitions

The **Defined Traps** tab is how you manage your existing trap definitions.



Event Name	OID	Category	Severity	Exec	Description	Active	Actions
abnormalCondition	.1.3.6.1.4.1.318.0.77	Status Events	SEVERE	/usr/local/bin/snmptraphandling.py "$r" "SNMP Traps" "$s" "$@" "$-*" "APC: An abnormal condition has been detected.: An abnormal condition has been detected."	SEVERE: An abnormal condition has been detected. The first variable is the fault condition. Variables: 1: mtrapargsInteger 2: mtrapargsString	Yes	
abnormalConditionCleared	.1.3.6.1.4.1.318.0.78	Status Events	INFORMATIONAL	/usr/local/bin/snmptraphandling.py "$r" "SNMP Traps" "$s" "$@" "$-*" "APC: An abnormal condition has been cleared.: An abnormal condition has been cleared."	INFORMATIONAL: An abnormal condition has been cleared. The first variable is the fault condition. Variables: 1: mtrapargsInteger 2: mtrapargsString	Yes	
accessViolationConsole	.1.3.6.1.4.1.318.0.46	Status Events	WARNING	/usr/local/bin/snmptraphandling.py "$r" "SNMP Traps" "$s" "$@" "$-*" "APC: Access violation via the console.: Three unsuccessful logins have been attempted via the console."	WARNING: Someone has attempted to login via the console with the incorrect password. Variables: 1: mtrapargsString	Yes	
accessViolationHTTP	.1.3.6.1.4.1.318.0.47	Status Events	WARNING	/usr/local/bin/snmptraphandling.py "$r" "SNMP Traps" "$s" "$@" "$-*" "APC: Access violation via HTTP.: An unsuccessful attempt to login via HTTP."	WARNING: Someone has attempted to login via HTTP with the incorrect password. Variables: 1: mtrapargsString	Yes	
airCriticalCondition	.1.3.6.1.4.1.318.0.306	Status Events	SEVERE	/usr/local/bin/snmptraphandling.py "$r" "SNMP Traps" "$s" "$@" "$-*" "APC Air: A critical condition was detected. : A critical condition was detected. "	SEVERE: An Air critical condition was detected. The first variable is the error condition text message. The second variable is the error number. Variables: 1: mtrapargsString02 2: mtrapargsInteger 3: mtrapargsString	Yes	

By default only 5 traps are displayed per page, this can be changed by using the ***x* Per Page** drop down list that appears on the right hand side of the table (top and bottom). The buttons either side of the page number count allow you to navigate back and forward through the pages.

You can jump to a specific page number by typing the number in the far right field and clicking the **Jump to Page** button.

When you have a large number of defined traps you can use the search field to find what you are after, this searches the **Event Name**, **OID** and **Description** fields.
The columns headings can be clicked on to sort the trap definitions as per your requirements.

	Event Name	OID	Category	Severity	Exec	Description	Active	Actions
☐	Areca	.1.3.6.1.4.1.18928.*	Status Events	CRITICAL	/usr/local /bin/snmptraphandling.py "ESXi Host Production 01" "SNMP Traps" "$s" "$@" "" "Areca RAID Controller Trap: $-*"		Yes	
☐	emsDeviceConfigChange	.1.3.6.1.4.1.318.0.252	Status Events	INFORMATIONAL	/usr/local /bin/snmptraphandling.py "$r" "SNMP Traps" "$s" "$@" "$-*" "APC: A device configuration change on a EMS.: A device configuration change has been made on a EMS."	INFORMATIONAL: A device configuration change has been made on the EMS. The first argument is the EMS serial number. The second argument is the EMS name. Variables: 1: emsIdentSerialNumber 2: emsIdentEMSName 3: mtrapargsString	Yes	

As explained earlier, you click the **edit** icon in the **Actions** column to edit a trap.

An individual trap can be duplicated by clicking the copy icon from the Actions column, this will prompt you to update data before copying and saving the trap.

An individual trap can be deleted by clicking the ✖ icon in the **Actions** column, this will prompt you before deleting the trap.

The **Active** column allows you to disable a trap definition without deleting it. When it is enabled, click the **Yes** word to disable it. This will turn to a **No**, to enable the trap click the **No** word.

You can **Delete**, **Enable** or **Disable** multiple trap definitions at once by using the left column field to select multiple traps and use the **With selected** controls.

Click **Go** to perform the requested action.

	Event Name	OID	Category	Severity	Exec	Description	Active	Actions
☑	apcDoorSenseConnected	.1.3.6.1.4.1.318.0.587	Status Events	INFORMATIONAL	/usr/local/bin/snmptraphandling.py "$r" "SNMP Traps" "$s" "$@" "$-*" "APC: A door sensor was connected.: A door sensor was connected."	INFORMATIONAL: A forced entry condition has been cleared. The first argument is the host device serial number. The second argument is the host device name. The third argument is the host device location. The fourth argument is the door identifier, (1=front, 2=rear). Variables: 1: mtrapargsString02 2: mtrapargsString03 3: mtrapargsString04 4: mtrapargsInteger 5: mtrapargsString	Yes	
☐	upsOverload	.1.3.6.1.4.1.318.0.2	Status Events	SEVERE	/usr/local/bin/snmptraphandling.py "$r" "SNMP Traps" "$s" "$@" "$-*" "APC UPS: Overload: The UPS has sensed a load greater than 100 percent of its rated capacity."	SEVERE: The UPS has sensed a load greater than 100 percent of its rated capacity. Variables: 1: mtrapargsString	Yes	
☑	apcDoorSenseDisconnected	.1.3.6.1.4.1.318.0.586	Status Events	SEVERE	/usr/local/bin/snmptraphandling.py "$r" "SNMP Traps" "$s" "$@" "$-*" "APC: A door sensor was disconnected.: A door sensor was disconnected."	SEVERE: A door sensor was disconnected. The first argument is the host device serial number. The second argument is the host device name. The third argument is the host device location. The fourth argument is the door identifier, (1=front, 2=rear). Variables: 1: mtrapargsString02 2: mtrapargsString03 3: mtrapargsString04 4: mtrapargsInteger 5: mtrapargsString	Yes	

With selected | Delete | Go

Delete
Enable
Disable

Page 1 of 1 | 5 Per Page

Managing Received Traps

The Received Traps tab is where you can report on all received traps that have been defined.

SNMP Trap Interface

SNMPTT is running

Received Traps | Add a Trap Definition | Defined Traps | Advanced

Showing records 1-7 of 7

Page 1 of 1 | 10 Per Page

	Timestamp	Event Name	OID	Trap Origin IP	Category	Severity	Actions
☐	2018-05-03 13:42:16	upsOnBattery	enterprises.318.0.5.0	10.25.254.50	Status Events	WARNING	✕
☐	2018-05-03 13:42:41	powerRestored	enterprises.318.0.9.0	10.25.254.50	Status Events	INFORMATIONAL	✕
☐	2018-05-03 13:43:03	upsOnBattery	enterprises.318.0.5.0	10.25.254.50	Status Events	WARNING	✕
☐	2018-05-03 13:43:34	powerRestored	enterprises.318.0.9.0	10.25.254.50	Status Events	INFORMATIONAL	✕
☐	2018-05-03 13:44:04	upsOnBattery	enterprises.318.0.5.0	10.25.254.50	Status Events	WARNING	✕
☐	2018-05-03 13:48:46	lowBattery	enterprises.318.0.7.0	10.25.254.50	Status Events	SEVERE	✕
☐	2018-05-03 13:50:17	powerRestored	enterprises.318.0.9.0	10.25.254.50	Status Events	INFORMATIONAL	✕

With selected | Delete | Go

Page 1 of 1 | 10 Per Page

The controls available on this page are almost identical to the Defined Traps tab. You can change the amount of records per page, search the records and delete (one or many).

Importing Trap Definitions

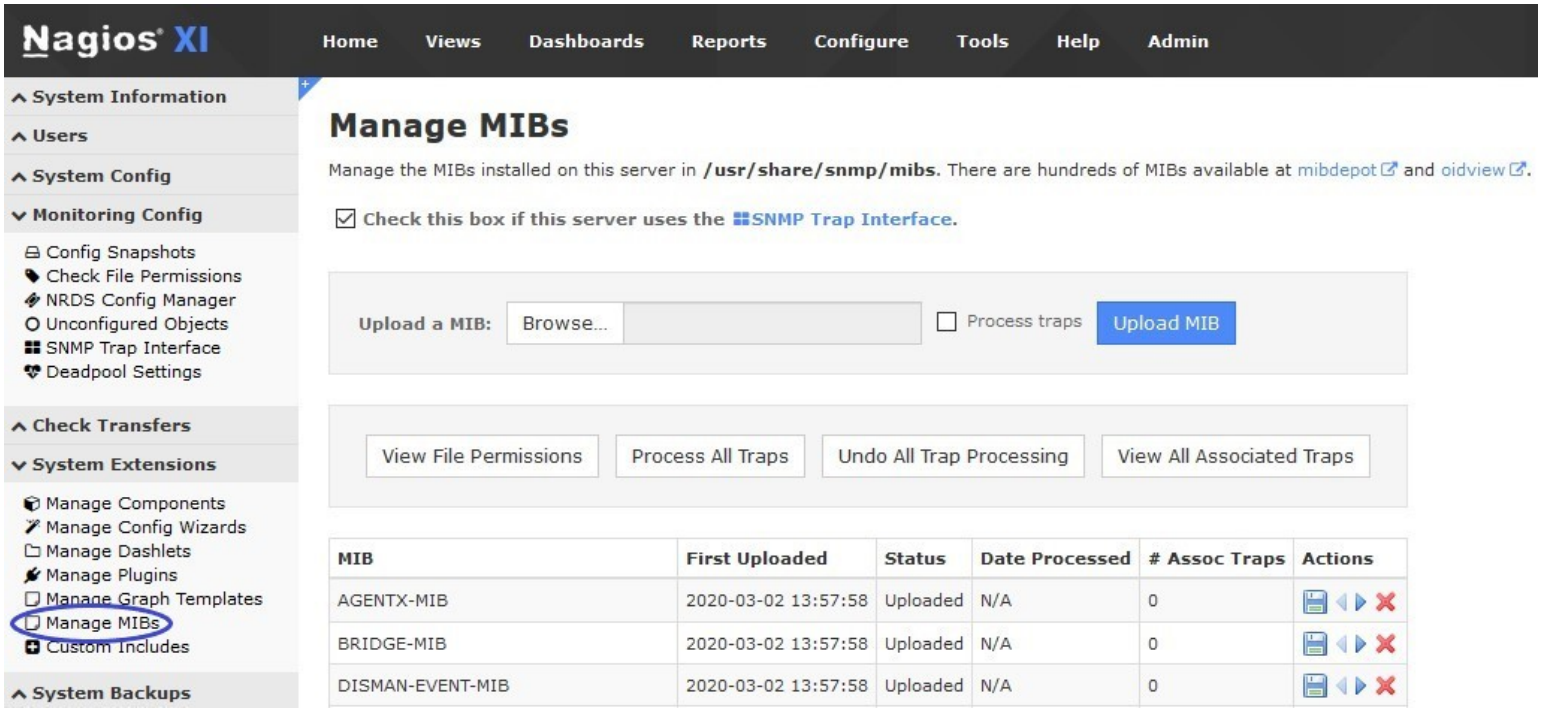
Its most likely that you will obtain a MIB file for your device that will include trap definitions, these can be imported into NXTI. There are several methods available for importing trap definitions into NXTI.

- Individually importing when uploading a MIB file
- Bulk import all MIB files with trap definitions
- Import existing trap definitions

Each method is explained in detail as follows.

Uploading A MIB File

You can upload a MIB file into Nagios XI via the **Admin** > **System Extensions** > **Manage MIBs** page.



First use the **Browse** button to locate the MIB file, select it, and click **Upload MIB**. Once complete you can navigate to NXTI and locate the newly imported MIBs.

XI lets you choose between two systems for handling traps.

1. Built-in SNMP trap interface
2. Legacy trap handling system in XI

If you wish to use the legacy system, you will need to deselect the **Check this box if this server uses the SNMP Trap Interface** check-box.

Process Individual Traps

You can process an individual trap checking the **Process traps** check-box prior to uploading the MIB. If you forgot to select the box – don't worry. You can still process the trap by clicking on the **Process Traps** actions button (right blue arrow).

To see which traps have been processed on a MIB-by-MIB basis, click on the number, shown in the **# Assoc Traps** column.

To undo trap processing on a MIB-by-MIB basis, click on the **Undo Trap Processing** actions button (left blue arrow).

You can download a MIB from your Nagios XI server to your workstation by clicking on the **Download** actions button (diskette icon).

You can also delete a MIB by clicking the **Delete** actions button (red X icon)

Process All Traps

You can process traps from all MIBs, installed on your system by pressing the **Process All Traps** button.

This will process each MIB file in the `/usr/share/snmp/mibs` directory, if a trap exists it will import it into NXTI. Additionally, it will also double check the `/etc/snmp/snmptt.conf` file to see if the trap is already defined. If it is defined then it will comment it out of the `snmptt.conf` file so there is no duplicate definition.

You can see all of the traps, processed so far by clicking **View All Associated Traps**.

You can undo all of the traps that have been processed by clicking **Undo All Trap Processing**.

Watch a video tutorial on how to upload and manage MIBs for SNMP in Nagios XI here:

https://support.nagios.com/kb/article/nagios-xi-uploading-and-managing-mibs-852.html

Import Existing Trap Definitions

Existing users of Nagios XI before the 5.5 release may already have traps configured in the `/etc/snmp/snmptt.conf` file. These can be imported into NXTI using a script which can save re-inventing the wheel. After they are imported the original `snmptt.conf` file needs to be truncated so there are no duplicate trap definitions. Follow these steps to import the `snmptt.conf` file into NXTI.

Establish a terminal session to your Nagios XI server as the root user. The first step is to change into the directory and create a backup of the `snmptt.conf` file by executing the following commands:

```
cd /etc/snmp/
cp snmptt.conf snmptt.conf.original
```

Now execute the following command to import the traps:

```
/usr/local/nagiosxi/scripts/nxti_import.php snmptt.conf
```

The script will output one line per trap it imports, here is some example output:

```
coldStart
warmStart
linkDown
linkUp
authenticationFailure
```

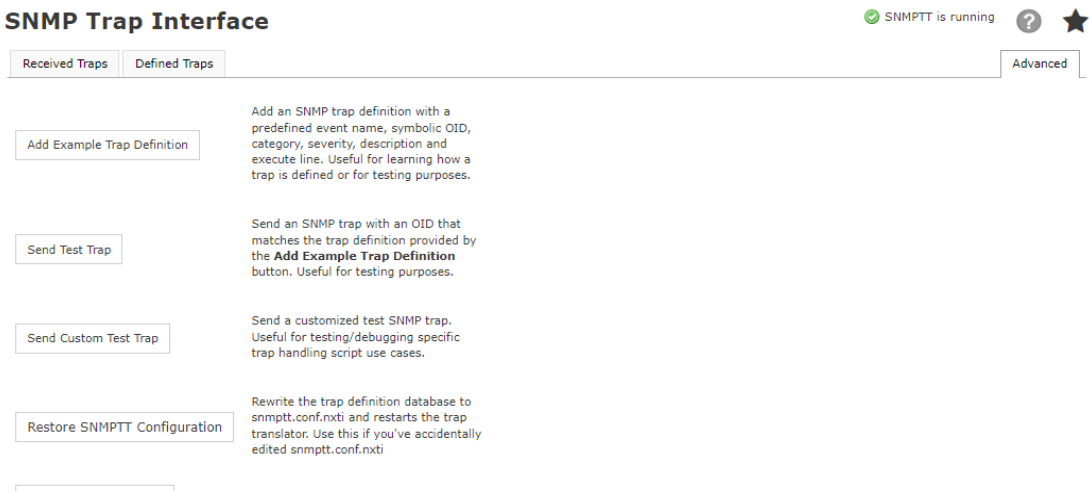
Now you need to truncate the `snmptt.conf` file by executing the following command:

```
echo '' > snmptt.conf
```

You can close the terminal session as you have completed this part of the import.
Next you need to open NXTI and check to make sure the traps have been imported, you'll find these under the **Defined Traps** tab.

The final step is to force the NXTI defined traps into the `snmptt.conf.nxti` file and restart the `snmptt` service. Both of these steps are performed by navigating to the **Advanced** tab and clicking the **Restore SNMPTT Configuration** button.

After completing these steps you will have successfully imported the existing traps in the `snmptt.conf` file into NXTI.



Advanced Features

The **Advanced** tab in NXTI provides various functionality as explained below.

- **Once Click Systems Test**
  - Performs a complete test of your local SNMPTT setup. It will add a trap definition, verify its existence in the database, send a matching snmptrap command, verify that the trap was received, check the output of the EXEC line, and then delete the definition, trap log, and output file.
- **Add Example Trap Definition**
  - Add an SNMP trap definition that is useful for learning or testing
- **Send Test Trap**
  - Allows you to send a trap that matches the example trap definition above
- **Send Custom Test Trap**
  - Allows you to send a custom test trap, a modal window appears with fields you can populate
- **Restore SNMPTT Configuration**
  - Pushes the settings from the NXTI configuration database back into the `/etc/snmp/snmptt.conf.nxti` file, useful if the file was accidentally edited
- **Show Test File Contents**
  - Displays a modal window with the contents of the `/usr/local/nagiosxi/var/NXTI_Write_Test` file
- **Show Unknown Trap Log**
  - Displays a modal window with the contents of the `/var/log/snmptt/snmpttunknown.log` file
  - This file can be useful for identifying received traps that yet do not have trap definitions created

Finishing Up

This completes the documentation on SNMP traps with NXTI.

If you have additional questions or other support related questions, please visit us at our Nagios Support Forums:

https://support.nagios.com/forum

The Nagios Support Knowledgebase is also a great support resource:

https://support.nagios.com/kb