Purpose

This document describes how to use event handlers in Nagios XI to take predefined actions when the hosts or services you are monitoring change state. Event handlers are used to automate processes taken when there is a state change for a specific host or service. This is useful because it reduces the amount of manual work when something changes in your environment. This document also includes some tips for advanced configurations and scripts.

Target Audience

This document is intended for use by Nagios XI administrators who want to implement custom event handler scripts beyond the simple stopping and restarting of services. Basic functional knowledge of shell scripting and Nagios macros is recommended.

Prerequisites

This document assumes that Nagios XI is installed and a few hosts and services are setup. Additionally, knowledge of the Core Config Manager (CCM) and an understanding of the general Nagios XI workflow will help as well.

What Is An Event Handler?

Event handlers are optional system commands (scripts or executables) that are run when a host or service state change occurs. These commands include, but are not limited to, restarting services, parsing logs, checking other host or service states, making database calls, etc. The possibilities are near limitless. Essentially, through event handlers, Nagios XI is capable of running any operation that can be performed from the command line with the added ability of utilizing macros passed by Nagios XI.

When Are Event Handlers Executed In Nagios XI?

Event handlers are called whenever a state change occurs. This includes HARD and SOFT state types, as well as OK, WARNING, CRITICAL, and UNKNOWN states. The logic for what actions to take, if any, when a state change occurs is performed by the script or executable that is called by Nagios XI at the moment of the state change. The script can parse these states through macros passed to it by the event handler.

For example, an event handler can be created that restarts a service if and only if that service has entered a HARD CRITICAL state. This is possible passing the script the `$SERVICESTATE$` (OK, WARNING, UNKNOWN, CRITICAL) macro and the `$SERVICESTATETYPE$` (HARD, SOFT) macro. It is the responsibility of the person writing the script to make sure the script logic handles the passed macros appropriately.

What Type Of Macros Can Be Passed By Event Handlers?

Any of the standard Nagios macros can be passed through an Event Handler to a script or binary. Usually the macros are relative to the to the Event Handler's service or host, though you can pass macros to your script that reference other hosts and services. For reference, a list of Nagios macros can be found at:

https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/macrolist.html

Minimum Requirements For A Working Event Handler

Event handlers have a few different parts:

- The check command itself within Nagios XI
- The event handler script called from the check command in Nagios XI

More complex event handlers can also pass macros to the event handler script and reference remote scripts.

Getting Started – Set Up Your First Event Handler In Nagios XI

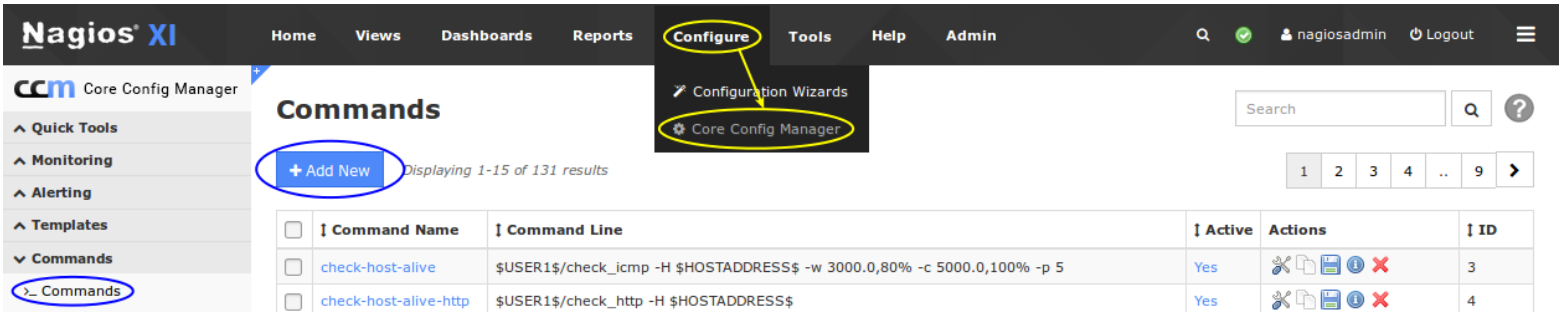
To keep things simple, bash scripts will be used for this document, even though events can run binaries, perl, python, php, and bash scripts among others. The first event handler will be a very simple bash script. It will

essentially write host information to a local text file when the host state changes. We will add functionality to this script as the document progresses. This example will follow these general steps:

I. Create a command in XI for the event handler
II. Create a dummy host for testing
III. Create a script in the libexec directory to be run by the handler
IV. Test the event handler
V. Add advanced features and macros

I. Create A Command In Nagios XI For The Event Handler

In the Nagios XI web interface Navigate to **Configure > Core Config Manager > Commands**.



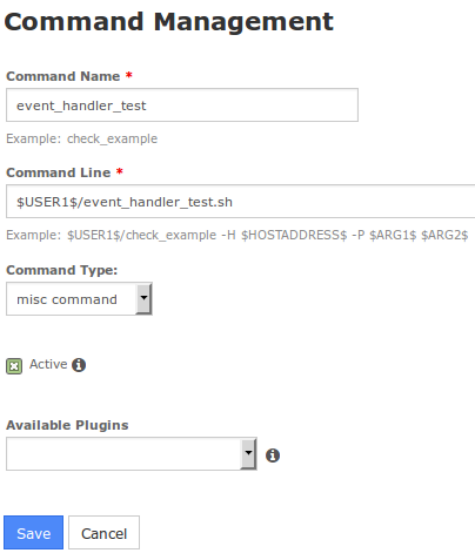
Click the **Add New** button and you will need to provide the following details:

Command Name: `event_handler_test`

Command Line: `$USER1$/event_handler_test.sh`

Command Type: `misc command`

Make sure the **Active** box is checked.



The final command definition should resemble the screenshot to the right.

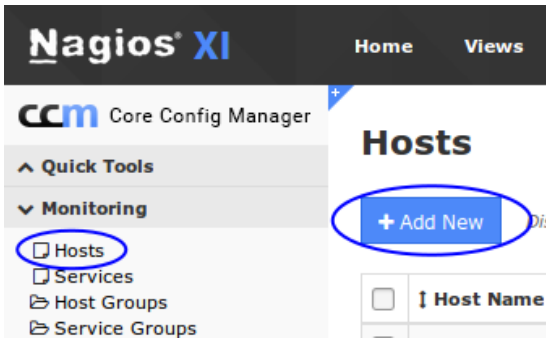
Click **Save** when finished.

`$USER1$` references the directory `/usr/local/nagios/libexec` from the `resource.cfg` file. This is the default path for plugins and scripts in Nagios XI. We will create a script later on to be used by this command, the name of the script will be `event_handler_test.sh`.

II. Create A Dummy Host

In CCM navigate to **Monitoring > Hosts**.

Click the **Add New** button and you will need to provide the following details:



Common Settings tab

Host Name: `event_handler_test`

Address: `127.0.0.1`

Check command: `check_dummy`

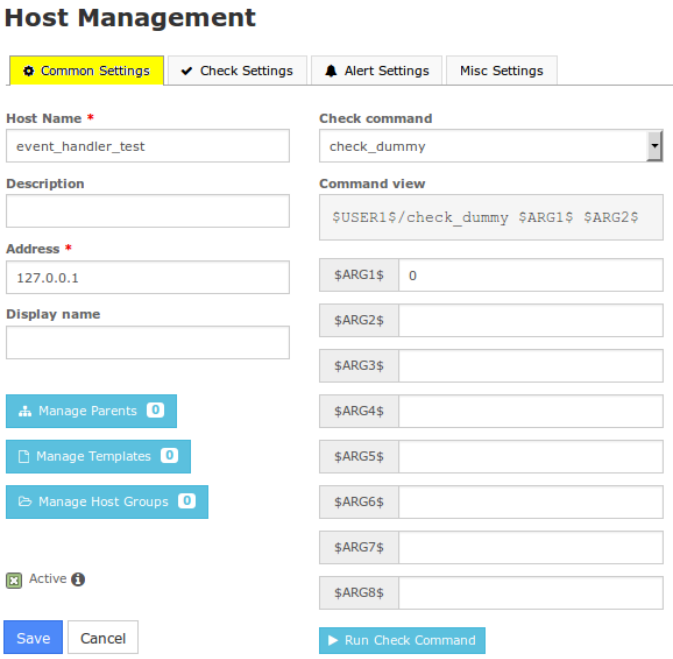
`$ARG1$: 0`

This forces the check_dummy command to always return 0 (UP)

Clicking Run **Check Command** should output **OK**.

This means you have set up the command correctly.

Check the **Active** checkbox.



Check Settings tab

Check interval: `5`

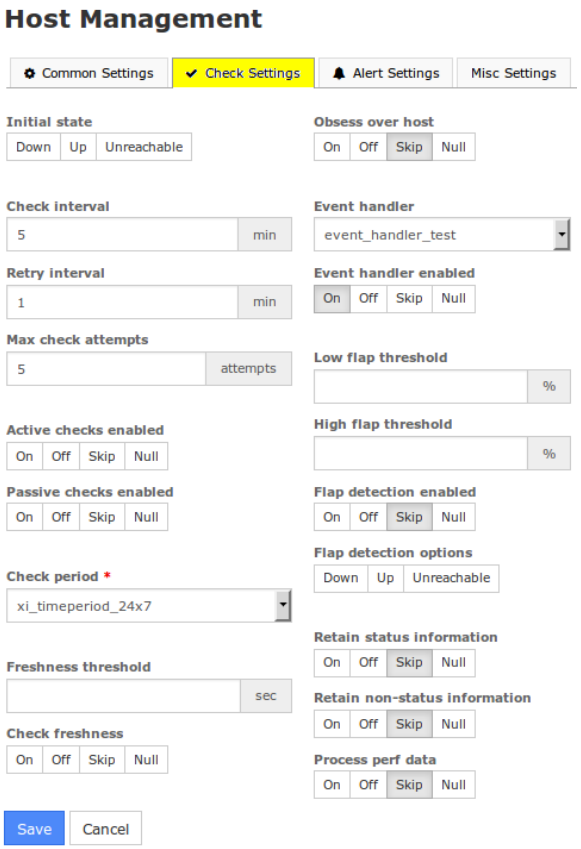
Retry interval: `1`

Max check attempts: `5`

Check period: `xi_timeperiod_24x7`

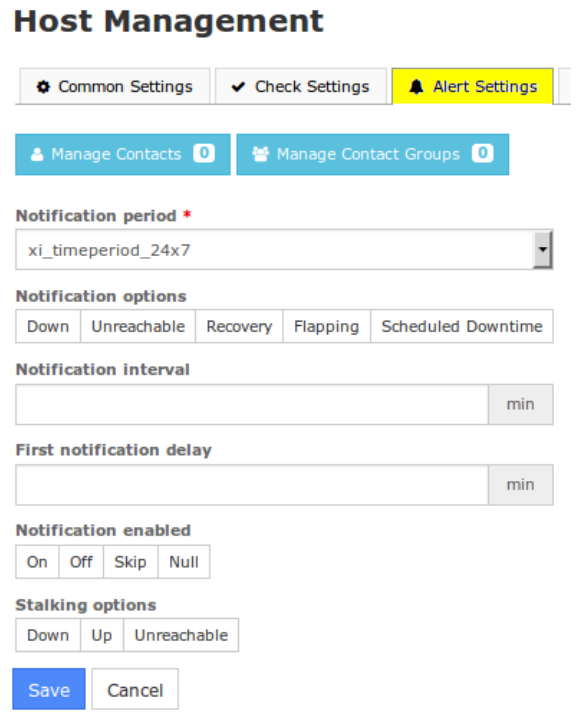
Event handler: `event_handler_test`

Event handler enabled: `On`



Alert Settings tab

Notification period: `xi_timeperiod_24x7`



Note: You can define the object's directives locally, on the host level (i.e. `notification_period`) or inherit the values from a host template (i.e. `notification_interval`, `notifications_enabled`, etc.) as it is shown in the picture.

Click the **Save** button to create the host.

Click the **Apply Configuration** button to apply the new configuration so the new `event_handler_test` host will become active.

III. Create A Script To Be Run By The Handler

Now that the Nagios XI event handler and host object are configured, we need to create a script to be run by the handler. The first iteration of the script will just output some very basic information to a text file in the `/tmp/` directory. In the proceeding sections of this document we will add functionality and complexities to the script and command of the handler (mostly macros and script logic). Before we get to the fun stuff, lets make sure the basic script works.

In all of the following steps you will be instructed to create and update the script. This will be done using the `vi` text editor. When using the `vi` editor:

To make changes press `i` on the keyboard first to enter insert mode

Make the required changes

Press `Esc` on the keyboard to exit insert mode

Type `:wq` to save the changes and quit the `vi` editor

Open up a terminal session to your Nagios XI server and log in as the root user. Execute the following commands:

```
cd /usr/local/nagios/libexec
touch event_handler_test.sh
chmod +x event_handler_test.sh
```

Now type the following command to edit the file with the `vi` editor:

```
vi event_handler_test.sh
```

Paste the following into the file:

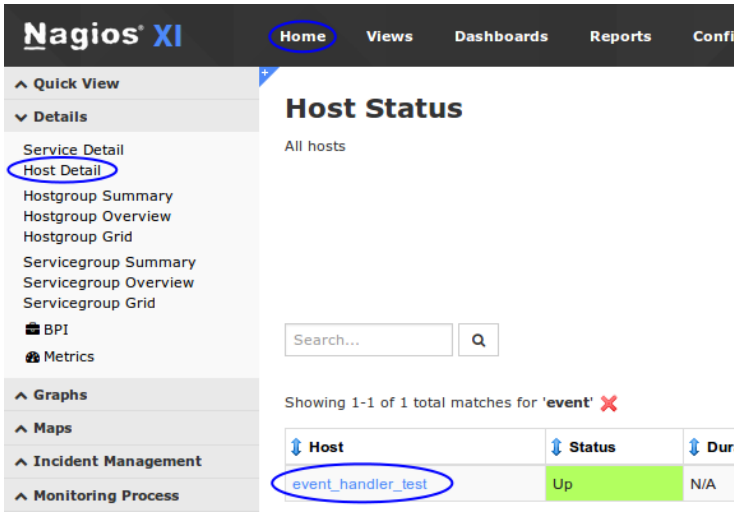
```
#!/bin/bash
DATE=$(date)
echo "The host has changed state at $DATE" > /tmp/hostinfo.txt
```

Save the changes.

IV. Test The Event Handler

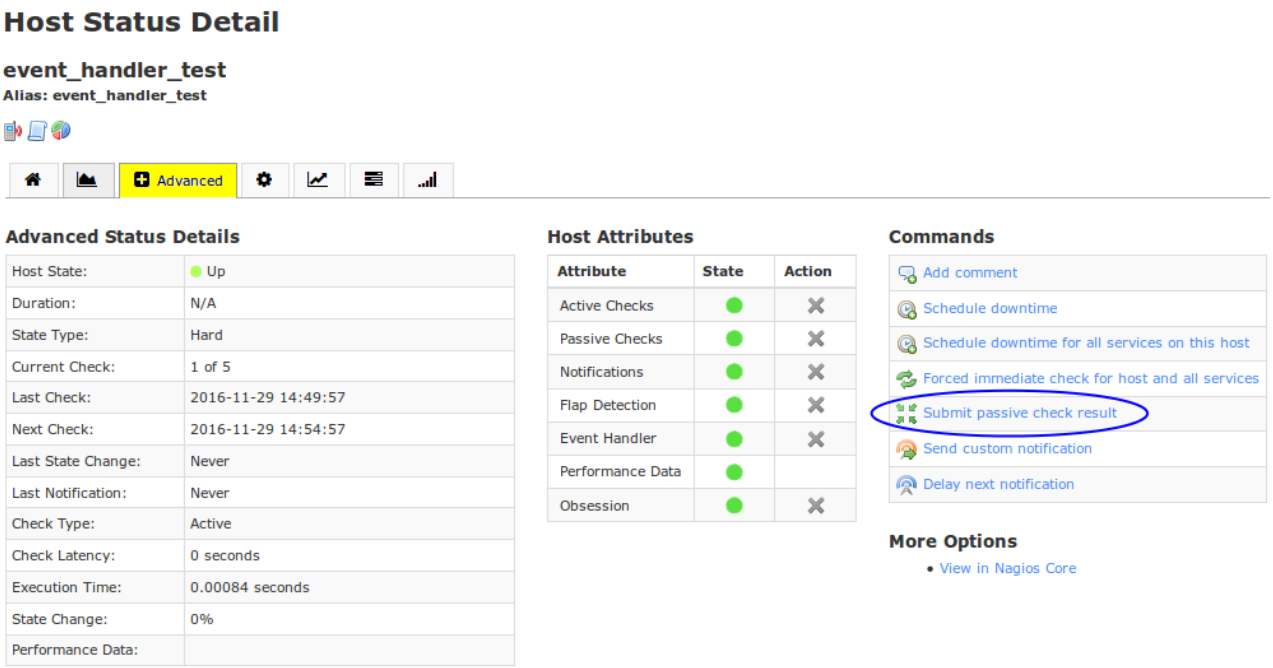
Now that everything is in place, we need to force the dummy host into a down state. By going into a down state the event handler will be triggered to run. This will create the file `/tmp/hostinfo.txt` as this is what the `event_handler_test.sh` script will do. Putting the host into a down state will be done by submitting a passive check with the **Check Result** of **DOWN**.

In the Nagios XI web interface navigate to **Home > Host Detail**.

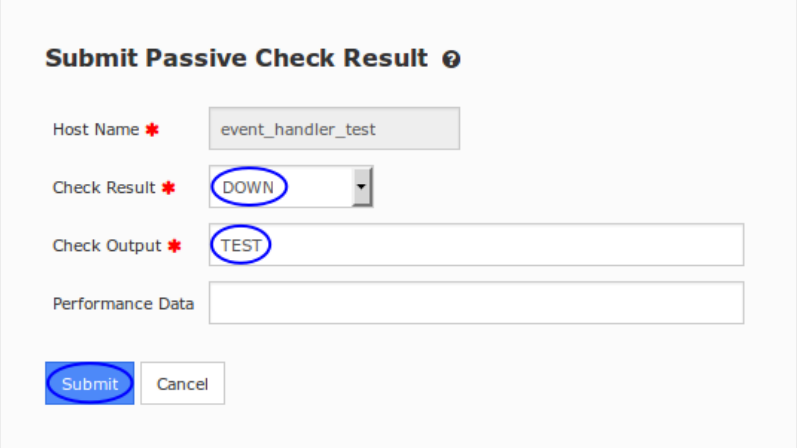


Click the host **event_handler_test**.

Click the **Advanced** tab.



Under the Commands table click **Submit passive check result**.



The click Submit passive check result will appear:

Check Result: **DOWN**
Check Output: **TEST**

Click **Submit**.

You should see the host change to a down state after a moment or two. Now lets check to see if the `/tmp/hostinfo.txt` file was created by running the following command from the Nagios XI server terminal session:

```
cat /tmp/hostinfo.txt
```

You should see output resembling:

```
The host has changed state at Tue Nov 29 15:10:01 AEDT 2016
```

Congratulations, you have created your first event handler. What this demonstrated is that Nagios executed the event handler when the host state changed.

These first sections really just scratch the surface, but it is important to understand all the parts of the event handler system before you attempt to implement some of the advanced features like passing macros and using them in script logic to perform tasks such as restarting services or executing other custom scripts. You can even use event handlers to pass commands back into Nagios through the command pipe. If you are unsure about the previous steps, take a moment to re-read them as things will get much more complicated from this point on.
Continue to the next section of the document to learn about the more advanced features of event handlers.

V. Advanced Features And Macros

Now that you have a fully functional event handler setup, you probably want to do something with it. The script we created in the previous steps does not really do much. The next step will show how macros work, as they are key to fully realizing the potential of event handlers. There are a number of macros that can be passed to the event handler script in Nagios, though they are object specific so some macros only work when passed from a service object, others from a host object, and yet others can be passed from both.

The event handler command definition needs to be edited to define which macros will be passed to the event handler script. In the next step we will add a few macros to the event_handler_test command and then add some logic to our script to print those macros to the `/tmp/hostinfo.txt` file. The macro list located on the Nagios core documentation website has a fairly comprehensive listing of all the Nagios macros available and which objects can pass them:

https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/macrolist.html

Adding Macros

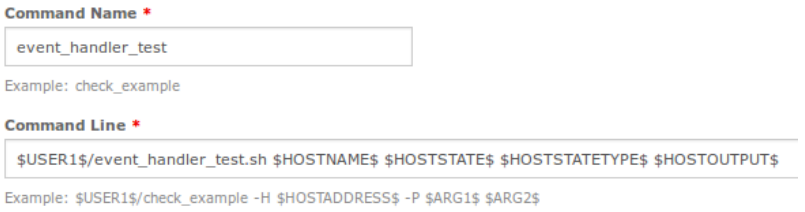
One thing to note is that you can only pass macros which are supported for your object type, and as we are working with a host check, we are limited to the host object macros. The most important macros that you will in all probability want to include in most, if not all your (host) event scripts, are:

- $HOSTNAME$
- $HOSTSTATE$
- $HOSTSTATETYPE$
- $HOSTOUTPUT$

In Nagios XI navigate to **Configure > Core Config Manager > Commands** and edit the `event_handler_test` command.

Append the above four macros, in order, to the **Command Line**, separating them with spaces.

When finished, **Save** and **Apply Configuration**.



Now the script needs updating. Return to the terminal session on the Nagios XI server and type the following command to edit the file with the `vi` editor:

```
vi event_handler_test.sh
```

Paste the following into the file:

```
#!/bin/bash
HOSTNAME=$1
HOSTSTATE=$2
HOSTSTATETYPE=$3
HOSTOUTPUT=$4
DATE=$(date)
echo "The host $HOSTNAME has changed to a $HOSTSTATETYPE $HOSTSTATE state at
$DATE with the error: $HOSTOUTPUT" > /tmp/hostinfo.txt
```

Save the changes.

Bash will receive the macros as arguments, enumerating them in order, giving them the names `$1`, `$2`, `$3` and `$4`. To make it easier to understand how they work in the script we will assign these to variables, using the original macro names.

As you can see from the changes, to the script, a variable is initially defined as a word (`HOSTNAME`), but it referenced by the echo command with a `$` sign (`$HOSTNAME`). You can see from the echo line how we are using the variables to output the data from the variables to the the `/tmp/hostinfo.txt` file.

Force the host into a down state once again by submitting a passive check result (as per the steps in part IV. Test The Event Handler). After you submit the passive check, wait a moment or two, and then check the contents of the `/tmp/hostinfo.txt` file:

```
cat /tmp/hostinfo.txt
```

Once again, you should see output resembling:

```
The host event_handler_test has changed to a HARD DOWN state at Tue Nov 29
16:48:57 AEDT 2016 with the error: TEST
```

If your output resembles the line above, congratulations! You can see the data from the macros are in bold to highlight that they were used. You now have a working event handler **with macros**. However, we are not finished yet, as this text file still does not do very much. We will need to add some logic to the script to check for certain strings in the macros and perform a task or two.

Adding Logic And Performing Tasks

The first bits of logic that most event handlers need is to separate out the different states of the host or service object. Additionally checking for a HARD or SOFT state is usually a good idea as most handlers are only needed when an object is in a HARD problem state. Most of the time your event handler will fire off an additional script that actually performs the tasks you are concerned about, sometimes passing additional arguments to that script.

On the next page is an updated version of the script that does the following:

- Assigns the received macro values to variables
- Assigns the current date to a variable
- Uses an **if** operator to determine if this is a SOFT state, if it is then the script exits as we don't want to do anything when it's in a SOFT state
- Uses a **case** statement to execute different commands depending if the `$HOSTSTATE` is UP, DOWN, or UNREACHABLE

- In each difference case statement you can see it is writing to a different file on the disk (`UP.txt`, `DOWN.txt`, or `UNREACHABLE.txt`). While this functionality is basic, it demonstrates the ability to perform different actions depending on the value of a macro that was received from Nagios.

```
#!/bin/bash
HOSTNAME=$1
HOSTSTATE=$2
HOSTSTATETYPE=$3
HOSTOUTPUT=$4
DATE=$(date)
if [ $HOSTSTATETYPE == 'SOFT' ]
then
     exit 0
fi
case "$HOSTSTATE" in
     UP)
     echo "The host $HOSTNAME changed to a $HOSTSTATE state @ $DATE" >
     /tmp/UP.txt
     ;;
     DOWN)
     echo "The host $HOSTNAME changed to a $HOSTSTATE state @ $DATE" >
     /tmp/DOWN.txt
     ;;
     UNREACHABLE)
     echo "The host $HOSTNAME changed to a $HOSTSTATE state @ $DATE" >
     /tmp/UNREACHABLE.txt
     ;;
esac
```

After making those changes try submitting some more passive check results so you can see how it is writing to different files on the disk.

Event Handler Script Use Case Example

Most often, event handlers just need to run one specific command for a certain state. This is the easiest of use cases to implement, especially if the command is meant to be run locally on the Nagios XI server. In the following example, we assume that the event handler is for the Nagios XI server, specifically for the yum update check. If updates are available, you want the event handler to run the command `yum update -y`.

Yum Updates Service Event Handler command line:

```
$USER1$/event_handler_yum.sh $SERVICESTATE$ $SERVICESTATETYPE$
```

The script event_handler_yum.sh is as follows:

```
#!/bin/bash
SERVICESTATE=$1
SERVICESTATETYPE=$2
if [ $SERVICESTATETYPE == 'SOFT' ]
     then exit
fi
case "$SERVICESTATE" in
OK)
     exit 0
     ;;
WARNING)
     yum update -y
     exit 0
     ;;
CRITCAL)
     yum update -y
```

```
        exit 0
        ;;
UNKNOWN)
        exit 2
        ;;
esac
```

If you noticed, the above macros are service specific, as this event handler is crafted for the yum updates service check. The macros not only changed, but the case string check did as well, as OK, WARNING, CRITICAL, and UNKNOWN are service states, not host states.

Running an External Script While Passing Macros

The other most common use case is to use the event handler to pass macros to another external script. This can include scripts that restart certain services, other Nagios plugin scripts (including `check_nrpe` or `check_by_ssh`), or even scripts to communicate to external ticketing systems. Event handlers are only limited by your imagination.

The event handler script below will assume that we want to pass macros to an external script that will prepare alerts for a ticketing system into the syntax that the ticketing system understands. The logic in the below script will only pass problems onto the ticketing system that are CRITICAL. Included in the arguments are a few more obscure macros, like `$SERVICEPROBLEMID$`, which is a unique number given every problem in Nagios. This is how our ticketing system will track the issues that are reported from Nagios. Like the previous example, this example will be a service event handler. If one wants to open tickets for host problem, another handler would have to be created, or the following script's logic extended.

Emailing to a Ticketing System:

Many Nagios XI administrators already maintain a separate ticketing system for managing technicians for outages. Integrating Nagios XI alerts with these systems usually is done through specially crafted emails. This can be done with an event handler script that sends a custom formatted email with the CLI email utility,

"sendmail". Those familiar with core will recognize the general format of sendmail command as it is similar to the core notification handlers. The following event handler and script are just a mock up and you will most definitely need to edit the format of the email for compatibility with your ticketing system.

Ticketing System Service Event Handler command line:

```
$USER1$/event_handler_service_ticket.sh "$HOSTNAME$" "$SERVICEDESC$"
$SERVICESTATE$ $SERVICESTATETYPE$ $SERVICEPROBLEMID$
```

The script `event_handler_ticket.sh` is as follows:

```
#!/bin/bash
DATE=$(date)
HOSTNAME="$1"
SERVICEDESC="$2"
SERVICESTATE=$3
SERVICESTATETYPE=$4
# SERVICEPROBLEMID and HOSTPROBLEMID are unique and are by far the best way to
track issues detected by Nagios
SERVICEPROBLEMID=$5
# Set your ticketing system's email below
EMAIL="email@domain.tld"
# The message below is a mock up.  You will need to edit the format to one
acceptable to your ticketing system
if [[ "$SERVICESTATETYPE" == "HARD"  && "$SERVICESTATE" == "CRITICAL" ]]; then
        /usr/bin/printf "%b" "***** Nagios Monitor XI Alert *****\n\nHost:
$HOSTNAME\nState:  $SERVICESTATE\n\nDate/Time: $DATE\n" | /bin/mail -s "OPEN
TICKET: $SERVICEPROBLEMID - $SERVICEDESC on $HOSTNAME is $SERVICESTATE " $EMAIL
        exit 0
else
        exit 0
fi
```

This script only submits tickets when the service in question is in a HARD, DOWN state. The portion of the `printf` command from "`%b`" to the pipe is the body of the email, while the string from "`/bin/mail -s`" to the `$EMAIL` macro is the subject. Most ticketing systems will look only at the subject of an incoming email to open a ticket, though some can also parse the body of the message.

Restarting a Service

Restarting a service from an event handler is covered in a pair of separate documents, with instructions for restarting Windows and Linux services. Please see the following links for more information on restarting a service with Nagios XI:

https://assets.nagios.com/downloads/nagiosxi/docs/Restarting-Windows-Services-With-NRPE-in-Nagios-XI.pdf
https://assets.nagios.com/downloads/nagiosxi/docs/Restarting_Linux_Services_With_NRPE.pdf

Other Notes, Options, And Additional Resources

The event handler system in Nagios XI is robust, extensible, and highly flexible. There are a few, more obscure and often overlooked options in Nagios XI that can be used to extend or alter the behavior of event handlers.

Is Volatile check attribute

This option is located at the bottom of the **Check Settings** tab for any service management page in CCM. Enabling this option will force Nagios XI to treat every check for the respective object as a state change. This will essentially run the event handler every time the object is checked, whether it is a scheduled active check, or a received passive check and it is state agnostic (the event handler will always run, no matter the object state). With this option enabled for an object with an event handler, Nagios XI can actually be used as a scheduler for routine maintenance or more specific tasks. More information can be found in the following link:

https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/volatileservices.html

Free variables

Sometimes the vanilla object macros are not enough for your scripting and event handler needs. Nagios XI includes the option to specify object or template specific variable to be used as macros. They are found on the **Misc Settings** tab for the object. Free variables must be prefixed by an underscore to avoid naming collision issues with the vanilla variables/macros. You can use these variables by treating them as macros to be passed to your event handler by referencing them as `$_YOURVARIABLE$` once defined on the object. See the following links for more information:

https://assets.nagios.com/downloads/nagiosxi/docs/Using-The-Nagios-XI-Core-Config-Manager-For-Host-Management.pdf
https://support.nagios.com/kb/article.php?id=539
https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/customobjectvars.html

On-Demand Macros (Cross-Object)

Complex event handler scripts occasionally need access to macros from other objects. The use cases include fail-over scenarios, elastic clusters, load-balancing, etc. Usually macros only reference variables that are values of the object in question. Occasionally, a developer will want to check the values of other object macros. For example, in a fail-over scenario, your event handler may want to check the status of a particular service on a different host other than the host that triggered the event.

If you would like to reference values for another host or service in an event handler, you can use what are called "on-demand" macros. On-demand macros look like normal macros, except for the fact that they contain an identifier for the host or service from which they should get their value.

Here's the basic format for on-demand macros:

```
$HOSTMACRONAME:host_name$
$SERVICEMACRONAME:host_name:service_description$
```

Note that the macro name is separated from the host or service identifier by a colon (:). For on-demand service macros, the service identifier consists of both a host name and a service description - these are separated by a colon (:) as well.

Examples of on-demand host and service macros follow:

```
$HOSTDOWNTIME:myhost$
$SERVICESTATEID:novellserver:DS Database$
$SERVICESTATEID::CPU Load$
```

More information about on-demand macros can be found at the following link:

https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/macros.html

BASH Scripting Tutorials

Although any programming/scripting language can be used, bash scripts are some of the easiest to create for simple tasks, do not require recompiling, but can still accomplish complex tasks if necessary. Throughout this document, all examples used were done in bash. If you are new to scripting, bash is a good place to start, especially as this document includes a number of script templates for ease of use with event handlers in Nagios XI. If you wish to learn more about bash scripting, you can put your favorite search engine to good use or check out the following links below:

http://www.tldp.org/LDP/Bash-Beginners-Guide/html/
https://help.ubuntu.com/community/Beginners/BashScripting

Finishing Up

This completes the documentation on how to use event handlers in Nagios XI to take predefined actions when the hosts or services you are monitoring change state.
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