Centreon

Poller States	Hosts	Up	Down	Unreachable	Pending	Services	Ok	Warning	Critical	Unknown	Pending
	208	207	1	0	0	3652	3342	11/24	72/206	70/80	0

Documentation - You are service.desk Logout

Monitoring Views Reporting

Services | Hosts | Event Logs

- By Status
  - Unhandled Problems
  - Service Problems
  - All Services
- By Host
  - Details
  - Summary
- By Host Group
  - Details
  - Summary
- By Service Group
  - Details
    - Problems
    - Acknowledged
    - Not Acknowledged
  - Summary
- Meta Services
  - Meta Services
- Nagios
  - Scheduling Queue
  - Downtime
  - Comments

Monitoring ▸ Services ▸ Details 2013/05/11 23:20

Nagios®
Understanding Macros and How They Work

Up To: Contents
See Also: List of Available Macros

Macros

One of the main features that make Nagios so flexible is the ability to use macros in command defintions. Macros allow you to reference information from hosts, services, and other sources in your commands.

Macro Substitution - How Macros Work

Before Nagios executes a command, it will replace any macros it finds in the command definition with their corresponding values. This macro substitution occurs for all types of commands that Nagios executes - host and service checks, notifications, event handlers, etc.

Certain macros may themselves contain other macros. These include the $HOSTNOTES$, $HOSTNOTESURL$, $HOSTACTIONURL$, $SERVICENOTES$, $SERVICENOTESURL$, and $SERVICEACTIONURL$ macros.

Example 1: Host Address Macro

When you use host and service macros in command definitions, they refer to values for the host or service for which the command is being run. Let's try an example. Assuming we are using a host definition and a *check_ping* command defined like this:

```
define host{
        host_name               linuxbox
        address         192.168.1.2
        check_command   check_ping
        ...
        }

define command{
        command_name    check_ping
        command_line    /usr/local/nagios/libexec/check_ping -H $HOSTADDRESS$ -w 100.0,90% -c 200.0,60%
        }
```

the expanded/final command line to be executed for the host's check command would look like this:

```
/usr/local/nagios/libexec/check_ping -H 192.168.1.2 -w 100.0,90% -c 200.0,60%
```

Pretty simple, right? The beauty in this is that you can use a single command definition to check an unlimited number of hosts. Each host can be checked with the same command definition because each host's address is automatically substituted in the command line before execution.

Example 2: Command Argument Macros

You can pass arguments to commands as well, which is quite handy if you'd like to keep your command definitions rather generic. Arguments are specified in the object (i.e. host or service) definition, by separating them from the command name with exclamation points (!) like so:

```
define service{
        host_name               linuxbox
        service_description     PING
        check_command   check_ping!200.0,80%!400.0,40%
        ...
        }
```

In the example above, the service check command has two arguments (which can be referenced with $ARGn$ macros). The $ARG1$ macro will be "200.0,80%" and $ARG2$ will be "400.0,40%" (both without quotes). Assuming we are using the host definition given earlier and a *check_ping* command defined like this:

```
define command{
        command_name    check_ping
        command_line    /usr/local/nagios/libexec/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$
        }
```

the expanded/final command line to be executed for the service's check command would look like this:

```
/usr/local/nagios/libexec/check_ping -H 192.168.1.2 -w 200.0,80% -c 400.0,40%
```

Tip: If you need to pass bang (!) characters in your command arguments, you can do so by escaping them with a backslash (\). If you need to include backslashes in your command arguments, they should also be escaped with a backslash.

On-Demand Macros

Normally when you use host and service macros in command definitions, they refer to values for the host or service for which the command is being run. For instance, if a host check command is being executed for a host named "linuxbox", all the standard host macros will refer to values for that host ("linuxbox").

If you would like to reference values for another host or service in a command (for which the command is not being run), 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:

- $*HOSTMACRONAME*:*host_name*$
- $*SERVICEMACRONAME*:*host_name*:*service_description*$

Replace *HOSTMACRONAME* and *SERVICEMACRONAME* with the name of one of the standard host of service macros found here.

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.


Tip: On-demand service macros can contain an empty host name field. In this case the name of the host associated with the service will automatically be used.

Examples of on-demand host and service macros follow:

```
$HOSTDOWNTIME:myhost$                          <--- On-demand host macro
$SERVICESTATEID:novellserver:DS Database$      <--- On-demand service macro
$SERVICESTATEID::CPU Load$                     <--- On-demand service macro with blank host name field
```

On-demand macros are also available for hostgroup, servicegroup, contact, and contactgroup macros. For example:

```
$CONTACTEMAIL:john$                            <--- On-demand contact macro
$CONTACTGROUPMEMBERS:linux-admins$             <--- On-demand contactgroup macro
$HOSTGROUPALIAS:linux-servers$                 <--- On-demand hostgroup macro
$SERVICEGROUPALIAS:DNS-Cluster$                <--- On-demand servicegroup macro
```

On-Demand Group Macros

You can obtain the values of a macro across all contacts, hosts, or services in a specific group by using a special format for your on-demand macro declaration. You do this by referencing a specific host group, service group, or contact group name in an on-demand macro, like so:

- $*HOSTMACRONAME*:*hostgroup_name*:*delimiter*$
- $*SERVICEMACRONAME*:*servicegroup_name*:*delimiter*$
- $*CONTACTMACRONAME*:*contactgroup_name*:*delimiter*$

Replace *HOSTMACRONAME*, *SERVICEMACRONAME*, and *CONTACTMACRONAME* with the name of one of the standard host, service, or contact macros found here. The delimiter you specify is used to separate macro values for each group member.

For example, the following macro will return a comma-separated list of host state ids for hosts that are members of the *hg1* hostgroup:

```
$HOSTSTATEID:hg1:,$
```

This macro definition will return something that looks like this:

```
0,2,1,1,0,0,2
```

Custom Variable Macros

Any custom object variables that you define in host, service, or contact definitions are also available as macros. Custom variable macros are named as follows:

- $_HOST*varname*$
- $_SERVICE*varname*$
- $_CONTACT*varname*$

Take the following host definition with a custom variable called "_MACADDRESS"...

```
define host{
        host_name               linuxbox
        address         192.168.1.1
        _MACADDRESS     00:01:02:03:04:05
        ...
        }
```

The _MACADDRESS custom variable would be available in a macro called $_HOSTMACADDRESS$. More information on custom object variables and how they can be used in macros can be found here.

Macro Cleansing

Some macros are stripped of potentially dangerous shell metacharacters before being substituted into commands to be executed. Which characters are stripped from the macros depends on the setting of the illegal_macro_output_chars directive. The following macros are stripped of potentially dangerous characters:

1. $HOSTOUTPUT$
2. $LONGHOSTOUTPUT$
3. $HOSTPERFDATA$
4. $HOSTACKAUTHOR$
5. $HOSTACKCOMMENT$
6. $SERVICEOUTPUT$
7. $LONGSERVICEOUTPUT$
8. $SERVICEPERFDATA$
9. $SERVICEACKAUTHOR$
10. $SERVICEACKCOMMENT$

Macros as Environment Variables

Most macros are made available as environment variables for easy reference by scripts or commands that are executed by Nagios. For purposes of security and sanity, $USERn$ and "on-demand" host and service macros are not made available as environment variables.

Environment variables that contain standard macros are named the same as their corresponding macro names (listed here), with "NAGIOS_" prepended to their names. For example, the $HOSTNAME$ macro would be available as an environment variable named "NAGIOS_HOSTNAME".

Available Macros

A list of all the macros that are available in Nagios, as well as a chart of when they can be used, can be found here.

Generated in 0.17 seconds