Abstract

This document aims to document the Monitoring Plugin Interface, a standard more or less strictly implemented by different network monitoring solutions. Implementers and Users of network monitoring solutions, monitoring plugins and libraries can use this as a reference point as to how these programs interface with each other.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-kaestle-monitoring-plugins-interface/.

Source for this draft and an issue tracker can be found at https://github.com/RincewindsHat/rfc-monitoring-plugins-interface.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 23 September 2024.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

Maintaining computer networks and providing services to machines, networks and humans is a complex task. Building infrastructures from the start commonly demands huge know-how in different technologies, but maintaining them afterwards is in many cases often more demanding. Complex systems can fail in a multitude of different ways, where the original problem might lead to symptoms which are often not immediately obvious and likely occur in a different place relative to the problem.

To ensure continuous and reliable operation the state and the functionality has to be monitored permanently with appropriate tools. These monitoring tools allow an operator, often a system and/or network administrator to read the state of the whole system (or specific subsystems) and detect problems when they occur.

The purpose of monitoring tools is therefore to determine the state of the systems in question and detect possible anomalies or problems,
measuring performance related metrics, process that data, produce a
human tangible representation and take action according to certain
rules, notify a system administrator for example. A system
implementing most or all of those tasks is called a "monitoring
system" in the following.

With the emergence of NetSaint/Nagios at the latest, a group of
network monitoring systems have relied on a loose group of programs
called "Monitoring Plugins" to do the lower level task of actually
determining the state of a particular entity or conduct measurements
of certain values.

The same interface was implemented by several different monitoring
solutions, including, without claiming completeness, Icinga, Icinga2,
Shinken, Naemon, Centreon and Opsview.

On the other side of this interface there are hundreds of individual
plugins developed by different people in different languages for a
multitude of purposes.

Examples for these are:

  * The monitoring plugins [https://www.monitoring-plugins.org/](https://www.monitoring-plugins.org/)
  * The nagios plugins [https://nagios-plugins.org/](https://nagios-plugins.org/)
  * Several monitoring plugins by Consol [https://labs.consol.de/de/nagios/](https://labs.consol.de/de/nagios/)
  * Several monitoring plugins by Linuxfabrik [https://github.com/Linuxfabrik/monitoring-plugins](https://github.com/Linuxfabrik/monitoring-plugins)
  * Several monitoring plugins by Davide Madrisan [https://github.com/Linuxfabrik/monitoring-plugins](https://github.com/Linuxfabrik/monitoring-plugins)

This document shall serve administrators of those monitoring systems
and especially developers of these monitoring plugins and monitoring
systems as a basis on how this interface should implemented, how the
plugins should work and how they should behave. It encourages the
standardization of libraries, monitoring plugins and monitoring
systems, to reduce the cognitive load on administrators and
developers, when they work with different implementations.
This document aims to be as general as possible and not to assume a special implementation detail, e.g. the programming language, the install mechanism or the monitoring system which executes the monitoring plugin.

1.1. Wording, Context and Scope

1.1.1. Wording

1.1.1.1. Monitoring system

A **monitoring system** is a collection of software components which serve the purpose of providing the system administrator of a particular system with an overview of the whole system. This ideally includes all of the devices, machines and components and their state as well as insights on particular components.

Most of the system mentioned here (for example Icinga, Naemon and Nagios) also provide a functionality to send notifications to the system administrator when something goes wrong, e.g. a particular component does not respond anymore or a certain threshold is exceeded.

For the purpose of this document a monitoring system is just "the thing that executes a monitoring plugin".

1.1.1.2. Monitoring plugin

A **monitoring plugin** is a standalone executable, which is executed by the monitoring systems to conduct one or multiple tests on behalf of the monitoring system.

The monitoring plugin does not rely on functionality provided by the monitoring system and is not a builtin of the monitoring system or linked against certain components of the monitoring system. Therefore it can also be executed manually and independently of a particular monitoring system.
The monitoring plugin can therefore be implemented independently of the monitoring system, it does not share necessarily share dependencies, the programming language or the distribution mechanism or other components with the monitoring system.

The monitoring plugin **may** accept parameters in the form of command line arguments, environment variables or configuration files (the location of which **may** in turn be given on the command line or via environment variable).

The monitoring plugin then proceeds to execute its duty and returns the result to the Monitoring System. Part of the process of returning the result is the termination of the execution of the Monitoring Plugin itself.

The execution of a monitoring plugin is typically short lived (in the order of seconds or milliseconds) and, while some implementations store some state on non-volatile memory to be able to reason over several execution cycles, this is not considered best practice. A monitoring plugin can not depend on any information other that which was given by the calling monitoring system, since the monitoring system might execute the plugin from a different system in the next cycle or switch between multiple systems.

A reasonable approach to thinking about monitoring plugins is to picture a "snapshot" of a current state, each execution independent from the others and wholly dependent on the input parameters and the "thing" that is to be monitored.

This "thing" which is to be monitored is not, in principle, restricted to any specific aspect of IT systems, apart from the restrictions above and the general concept. Examples for areas which are difficult to cover with this approach, are statistical analyses of time series data or event monitoring, such as log monitoring. However querying system, which are collecting and processing this kind of data, would be a valid indirection.

A popular example for this behaviour is the extraction of bandwidth usage on most switches. Commonly this is only exposed as counters for each network interface, one for outgoing and one for incoming bytes. The absolute value of bytes is practically useless without knowing the value which was read previously and the time difference between the probes. In this scenario, different workaround are possible, if the device itself does not provide the rate values:

*The monitoring plugin queries the values several times during its execution cycle with a know time difference between the queries, which allows rate calculation in this (typically) short time*
frame. Nothing is known effectively about the time between execution cycles.

*The monitoring system executes the monitoring plugin with the data and date of the last execution as parameter, which allows for proper rate calculation.

*A system queries the devices regularly for the absolute values and stores them. The monitoring plugin then queries this system for the collected values (and timestamps) or directly for a statistical analysis

1.1.2. Scope

The scope of this document is limited to the interaction of a monitoring system and a monitoring plugin, meaning the interface connecting them.

It does not attempt to describe the inner workings of a specific implementation of either monitoring system or monitoring plugin.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Range expressions

In many cases thresholds for metrics mark a certain range of values where the values are considered to be good or bad depending on whether they are inside or outside of this range. For a significant number of metrics an upper (e.g. load on unixoid systems) or lower (e.g. effective throughput, free space in memory or storage) border might suffice, for some it does not, for example a temperature value from a temperature sensor should be within certain range (e.g. between 10°C and 45°C).

Regarding input parameters this might be handled with options like --critical-upper-temperature and --critical-lower-temperature, but this creates a problem in the performance data output, if only scalar values could be used. To resolve this situation the Range expression format was introduced, with the following definition:
range-expression = [direction-switch] bounds

bounds = (lower-bound / upper-bound) / lower-bound upper-bound

direction-switch = "@"

lower-bound = NUMERAL ":" 

upper-bound = NUMERAL

NUMERAL = ["-"|1*DIGIT [ "."|1*DIGIT ]] ; numerical value, either integer or floating point

where:

1. At least start or end MUST be provided.
2. start <= end
3. If start == 0, then start can be omitted.
4. If end is omitted, it has the "value" of positive infinity.
5. Negative infinity can be specified with the tilde character ~.
6. If the prefix @ IS given, the value exceeds the threshold if it is INSIDE the range between start and end (including the endpoints).
7. If the prefix @ is NOT given, the value exceeds the threshold if it is OUTSIDE of the range between start and end (including the endpoints).

2.1.1. Examples

<table>
<thead>
<tr>
<th>Range definition</th>
<th>Exceeds threshold if x...</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt; 0 or &gt; 10, (outside the range of {0 .. 10})</td>
</tr>
<tr>
<td>10:</td>
<td>&lt; 10, (outside {10 .. ∞})</td>
</tr>
<tr>
<td>~:10</td>
<td>&gt; 10, (outside the range of {−∞ .. 10})</td>
</tr>
<tr>
<td>10:20</td>
<td>&lt; 10 or &gt; 20, (outside the range of {10 .. 20})</td>
</tr>
<tr>
<td>@10:20</td>
<td>≥ 10 and ≤ 20, (inside the range of {10 .. 20})</td>
</tr>
</tbody>
</table>

Table 1

3. Input Parameters for a Monitoring Plugin

A Monitoring Plugin MUST expect input parameters as arguments during execution, if any are needed/expected at all. It MAY accept these parameters given as environment variables and it MAY accept them in a configuration file (with a default path or a path given via arguments or environment variables).
In general positional arguments are strongly discouraged.

Some arguments **MUST** have this predetermined meaning, if they are used:

<table>
<thead>
<tr>
<th>Argument (long)</th>
<th>Argument (short version, optional)</th>
<th>Argument</th>
<th>Meaning</th>
<th>optional</th>
<th>can be given multiple times</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help</td>
<td>-h</td>
<td>None</td>
<td>Triggers the help functionality of the Monitoring Plugin, showing the individual parameters and their meaning, examples for usage of the Monitoring Plugin and general remarks about the how and why of the Monitoring Plugin. <strong>SHOULD</strong> overwrite all other options, meaning, they are ignored if --help is given. The Monitoring Plugin <strong>SHOULD</strong> exit with state UNKNOWN (3).</td>
<td>no</td>
<td>-- (makes no difference)</td>
</tr>
<tr>
<td>--version</td>
<td>-V</td>
<td>None</td>
<td>Shows the version of the Monitoring Plugin to allow users</td>
<td>no</td>
<td>-- (makes no difference)</td>
</tr>
<tr>
<td>Argument (long)</td>
<td>Argument (short version, optional)</td>
<td>Argument</td>
<td>Meaning</td>
<td>optional</td>
<td>can be given multiple times</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>--timeout</td>
<td>-t</td>
<td>Integer (meaning seconds) or a time duration string</td>
<td>to report errors better and therefore help them and the developers. The Monitoring Plugin <strong>SHOULD</strong> exit with state UNKNOWN (3).</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Sets a limit for the time which a Monitoring Plugin is given to execute. This is there to enforce the abortion of the test and improve the reaction time of the Monitoring System (e.g. in bad network conditions it might be helpful to abort the test prematurely and inform the user about that, than trying forever to do something which won't succeed. Or if soft real
<table>
<thead>
<tr>
<th>Argument (long)</th>
<th>Argument (short version, optional)</th>
<th>Argument</th>
<th>Meaning</th>
<th>optional</th>
<th>can be given multiple times</th>
</tr>
</thead>
<tbody>
<tr>
<td>--hostname</td>
<td>-H</td>
<td>String, meaning either a DNS name or an IP address of the targeted system</td>
<td>If the Monitoring Plugin targets exactly one other system on the</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Argument (long)</td>
<td>Argument (short version, optional)</td>
<td>Argument</td>
<td>Meaning</td>
<td>optional</td>
<td>can be given multiple times</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>--verbose</td>
<td>-v</td>
<td>None</td>
<td>Increases the verbosity of the output, thereby breaking the suggested rules about a short and concise output. The intention is to provide more information to a user.</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>--exit-ok</td>
<td></td>
<td></td>
<td>The Monitoring Plugin exits unconditionally with OK (0). Mostly useful for the purpose of packaging and testing plugins, but might be used to always ignore errors (e.g. to just collect data).</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
3.1. Examples

For the execution with --help:

$ my_check_plugin --help

the output might look like this:

my_check_plugin version 3.1.4
Licensed under the AGPLv1.
Repository: git.example.com/jdoe/my_check_plugin

This plugin just says hello. It fails if you don't give it a name.

Usage:
my_check_plugin --name NAME [--greeting GREETING]

Options:
--help
  this help
--version
  Shows the version of the plugin
--name NAME
  if given, uses NAME as a name to greet.
--greeting GREETING
  if given, uses GREETING instead of Hello.

Examples:
$ my_check_plugin --name Jane
Hello Jane

$ my_check_plugin --greeting Ciao --name Alice
Ciao Alice

This imaginary Monitoring Plugin tries to be really helpful here, displays the version, the license and the upstream repository with the help (although not necessary), has a short description about the purpose, lists the options in an easily readable way and even gives some examples.

For the execution with --version

$ my_check_plugin --version

the output might be a bit shorter:

my_check_plugin version 3.1.4
or even:

3.1.4

where both show the necessary information.

4. Output of a Monitoring Plugin

The output of a Monitoring Plugin consists of two parts on the first level, the Exit Code and output in textual form on stdout.

4.1. Exit Code

The Monitoring Plugin **MUST** make use of the Exit Code as a method to communicate a result to the Monitoring System. Since the Exit Code is more or less standardized over different systems as an integer number with a width of or greater than 8bit, the following mapping is used:

<table>
<thead>
<tr>
<th>Exit Code (numerical)</th>
<th>Meaning (short)</th>
<th>Meaning (extended)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
<td>The execution of the Monitoring Plugin proceeded as planned and the tests appeared to function properly and the measured values are within their respective thresholds</td>
</tr>
<tr>
<td>1</td>
<td>WARNING</td>
<td>The execution of the Monitoring Plugin proceeded as planned and the tests appeared to <strong>not</strong> function properly or the measured values are <strong>not</strong> with their respective thresholds. The problem(s) do(es) <strong>not</strong> seem exceptionally grave though and do(es) <strong>not</strong> require immediate attention</td>
</tr>
<tr>
<td>2</td>
<td>CRITICAL</td>
<td>The execution of the Monitoring Plugin proceeded as planned and the tests appeared to <strong>not</strong> function properly or the measured values are <strong>not</strong> with their respective thresholds. The problem(s) do(es) <strong>seem</strong> exceptionally grave though and do(es) <strong>require</strong> immediate attention</td>
</tr>
<tr>
<td>3</td>
<td>UNKNOWN</td>
<td>The execution of the Monitoring Plugin <strong>did not</strong> proceed as planned. The reasons might be manifold, e.g. missing permissions, missing libraries, no available network connection to the destination, etc.. In summary: The Monitoring Plugin could <strong>not</strong> determine the state of whatever it should have been checking and can therefore make no reliable statement about it.</td>
</tr>
<tr>
<td>4-125</td>
<td>reserved for</td>
<td></td>
</tr>
</tbody>
</table>
Exit Code (numerical) | Meaning (short) | Meaning (extended)
---|---|---
future use |

Table 3

4.2. Textual Output

The textual output should consist of printable characters end of this output is marked by EOF. There is no length limitation per se, but it a limit of 512kiB would be reasonable and should not be exceeded to avoid influencing the performance of the monitoring system, also some system might limit the output arbitrarily.

The original purpose of the output on stdout was to provide information for the user of the Monitoring System in a free text form; a way for the Monitoring Plugin to communicate further details on what happened and what the current state is. This purpose still exists, but was expanded with the, so called, performance data to allow the machine readable communication of measured values for further processing in the Monitoring System, e.g. for the creation of diagrams.

Therefore the further explanation is split into free form output and performance data. The general schema is the following:
4.2.1. Free form output

This part of the output should give an user information about the state of the test and, in the case of problems, ideally hint what the origin of the problem might be or what the symptoms are. If the test relies on numeric values, this might be displayed to give an user more information about the specific problem. It might consist of one or more lines (separated by CRLF or LF) of unicode symbols. Considering the age and implementation of current systems, restricting the output to US-ASCII characters is a safe choice.

Although no strict guidelines for creating this part of the output can really be given due to its free form character, a developer should keep a potential reader in mind. It might, for example, be OK to put the output in a single line if there are only one or two items of a similar type (think: multiple file systems, multiple sensors, etc.) are present, but not if there 10 or 100, although this might present a valid use case. If there are several different items exists in the output of the Monitoring Plugin they probably SHOULD be given their own line in the output.

The free form part is not intended for deep diagnostics, logging or too detailed reports, therefore it should be kept rather short.
4.2.1.1. Examples

Remaining space on filesystem "/" is OK

Sensor temperature is within thresholds

Available Memory is too low

Sensor temperature exceeds thresholds

    are OK, but

Remaining space on filesystem "/" is OK ( 62GiB / 128GiB )

Sensor temperature is within thresholds ( 42°C )

Available Memory is too low ( 126MiB / 32GiB )

Sensor temperature exceeds thresholds ( 78°C > 70°C )

    are better.

4.2.2. Performance data

In addition to the free form part the output can contain machine readable measurement values.

In addition to the format definition earlier, the following contains some constraints and best practices:

1. label MUST consist of at least one non-space character, but can otherwise contain any printable characters except for the equals sign (=) or single quotes ('). If it contains spaces, it must be surrounded by single quotes

2. value is a numerical value, might be either an integer or a floating point number. Using floating point numbers if the value is really discreet SHOULD be avoided. The representation of a floating point number SHOULD NOT use the "scientific notation" (e.g. 6.02e23 or -3e-45), since some systems might not be able to parse them correctly. Values with a base other then 10 SHOULD be avoided (see below for more information on Byte values).

3. UOM is the Unit of measurement (e.g. "B" for Bytes, "s" for seconds) which gives more context to the Monitoring System.

*The following constraints MUST be applied:

1. An UOM of % MUST be used for percentage values
2. An UOM of c **MUST** be used for continuous counters (commonly used for the sum of bytes transmitted on an interface)

*The following recommendations **SHOULD** be applied:

1. The UOM for Byte values is B and although many systems do understand units like KB, KiB, MB, GB, TB they **SHOULD** be avoided, at the least to avoid the ugly hassle about people misinterpreting the base10 values as base2 values and the other way round. This is also a prime example where floating point number **SHOULD NOT** be used, since there are obviously only integer numbers included.

2. The UOM for time is s, meaning seconds, SI-Prefixes (e.g. ms for milli seconds) are allowed if necessary or useful.

3. In general, SI units and SI prefixes **MAY** be used as UOM if applicable, but the Monitoring System may not understand them correctly (mostly in uncommon cases), in that cases appropriate workarounds **MAY** be applied on the side of the Monitoring Plugin. Since the values are not intended to be human readable, normalized units are recommended (e.g. overall_power=1400000000W instead of overall_power=14GW)

4. warn and crit are the threshold values for this measurement, which may have been given by the user as input, may be hardcoded in the Monitoring Plugin or may be retrieved from a file or a device or somewhere else during the execution of the Monitoring Plugin. The unit used **MUST** be the same as for value. These values are not simple numbers, but range expressions (Section 2.1).

5. min and max are the minimal respectively the maximal value the value could possibly be. The unit **MUST** be the same as for value. These values can be omitted, if the value is a percentage value, since min and max are always 0 and 100 in this case.

5. **Implementation Status**

The interface mentioned here is implemented by several network monitoring systems. A non-exhaustive list of these systems includes:

* Icinga 2

* Naemon
Nagios

The other side of the interface is implemented by several different projects, again in a non-exhaustive list:

*The Monitoring Plugins Project

*The Nagios Plugins Project

*The Linuxfabrik Monitoring Plugins

*Madrisan Nagios Plugins

6. Security Considerations

Special security considerations are hard to define regarding this topic. Regarding the implementation of this interface, the usual programming security considerations should apply (e.g. sanitize inputs), but the risks and problems regarding security are dependent on the specific implementation and usage.

7. IANA Considerations

This document has no IANA actions.

8. Normative References


Acknowledgments

Thanks for previous have to be said to the original inventors of this interface, although it is not easy to determine who these persons are, so they are mentioned here in general.

Thanks are going also to the many different implementors on either side of this interface for their hard work which allows the use of different components and systems with each other in the best spirit of free software.

Author's Address

Lorenz Kästle
The Monitoring Plugins Project

Email: lorenz@vulgrim.de