Cheatography

Guidelines for writing checks for Check_MK Cheat Sheet

by dlicudi via cheatography.com/22839/cs/4821/

Naming

☑ The check types should be named short and unique. They must consist only of lower case characters, digits and underscores and begin with a lower case character.

☑ Checks where one item of the check represents one thing (e.g. fan, power supply), should be named in singular, e.g. `casa_fan`, `if`, `oracle_tablespace`. Checks where each item checks a quantity, e.g. number of logins, should be named in plural (e.g. `user_logins`, `printer_pages`). Note: due to historic misconducts many existing check types are named contrarily to this rule. That does not mean that new checks should be named inconsistently as well!

☑ *Vendor specific checks* must be prefixed with a vendor specific unique abbreviation (which you think of). Example: `fsc_` for Fujitsu Siemens Computers.

☑ *Product specific checks* must be prefixed with a product abbreviation, for example `steelhead_status` for a Steelhead appliance of Riverbed.

☑ *SNMP based checks*: if the check makes use of a standardized MIB which is or might be implemented by more than one vendor, then the check should not be named after the vendor but after the MIB. An example are the hr_* checks.

☑ Service descriptions of different check types fundamentally doing the same must be identical (e.g. `if`/`if64`/`ifoperstatus`). Reason: this makes rules in `main.mk` simpler for the user!

Configuration variables

☑ Configuration variables for `main.mk` should be named after the check if they are only used by this check. This does not hold for variables, that are used by several checks (e.g. `filesystem_default_levels` is used by `df`, `hr_fs`, `df_netapp`, ...)

Configuration variables (cont)

☑ The variable that is used for the check's default parameters and entered in the inventory function must be named CHECKTYP`_default_levels` (if not used by more than one check, see above). Example: check foo_bar has the configuration variable `foo_bar_default_levels`.

☑ If a check does not use check parameters, the inventory function must return None as parameter and the check function must name the parameter argument `_no_params`.

☑ The name of the inventory and check function must be prefixed with the name of the check type, for example `inventory_h3c_lanswitch_cpu` for the `check h3c_lanswitch`.

Plugin output

Each check returns one line of text - the plugin output (or sometimes called check output). In order to unify things the output must be formated according to the following rules:

☑ when returning measurement values, place exactly one space between the value an the unit (e.g. 17.3 V). Only exception: Put **no** space before a percent sign. (correct e.g. 89.4%).

☑ When returning measurement values, name the names of the quantities in upper case, then add the value separated by a colon. Examples: Voltage: `24.5 V`, Phase: `negative`, Flux-Capacitor: `operational`

☑ Do not directly use return codes or cryptic return strings internal to the device. Instead, try to translate them to human readable messages. Example: Instead of `routeMonitorFail` use `route monitor has failed`

Performance data

Format of Performance data

☑ Always send int or float data as performance data. Do not attach a unit. Write temp instead of "%0.2fC" % temp!

☑ If you need to omit fields in the middle of the data list (e.g. warn or crit), add a None instead, for example [("usage", usage, None, None, 0, size)]

☑ If you need to omit fields at the end, simply omit them. Do not add trailing Nones.

☑ Naming of performance data variables: Names consist of only lowercase letters and underscores (rare). Also trailing digits are allowed (e.g. phase3).

☑ Naming of performance data variables: The name of the variable should be named correctly after the thing, not after the unit. Example: use current instead of ampere. Use size instead of bytes.

Always use the canonical unit: send Bytes, not KB, MB or GB. Send Celsius, not Fahrenheit. Send Bits/sec, not MBits/sec. It is the task of the graphing tool to do a useful scaling.

Performance data flag

☑ Only set "has_perfdata" to True in check_info if the check really produces performance data output.

PNP Graph definition

☑ Each check returning performance data must have a dedicated PNP graph definition in pnp-templates. If the check has warning and critical levels, the graph must display these levels as yellow and red lines.

☑ PNP graphs should always use the consolidation function MAX (there are some rare exceptions where only MIN makes sense).

By **dlicudi**

cheatography.com/dlicudi/

Published 11th August, 2015.

Last updated 11th August, 2015.

Performance data (cont)

☑ However: the Average value which is printed in the labelling of the graph must use the consolidation function AVERAGE. Using MAX would compute the average of the maximum values - which is totally useless.

RRA definition

☑ Each check returning performance data must also have an RRA definition specifiying which of MAX, MIN and AVERAGE is needed to display the graph in its current (and maybe future) forms. These definitions are in pnp-rraconf. Use a symlink here.

Perf-O-Meter

☑ Each check returning performance data should have a Perf-O-Meter. For checks which are part of Check_MK the Perf-O-Meter must be defined in web/plugins/perfometer/check_mk.py. For third-party checks it should be defined in a separate file in web/plugins/perfometer.

SNMP based checks

☑ Only use numeric OIDs in your checks. Name-based OIDs rely on MIB files and the check won't work when the MIB files are not in place. Always have your OIDs start with a root, for example: .1.3.6.1.4.1

Simple memory checks

Many devices report memory usage in a simple way: used and total memory in absolute terms, or, equivalently, used and free memory in absolute terms.

☑ To ensure uniform behaviour, all these checks should use the check_memory function defined in memory.include.

☑ The check group should be memory_simple. Note that this requires that the check has an item. For devices with no modules, (i.e. only one memory value) the item should be the empty string.

☑ The service description should be "Memory" or "Memory %s" for checks with nonempty items.

Check Layout

☑ All checks must follow the same layout specified below:

- fileheader with GPL notice
- name and email address of the author - if check was contributed
- example output as sent by the agent
- default settings of configuration variables
- helper functions and variables, if any are needed
- the inventory function
- the check function
- the `check_info` declaration

Coding Style: Add an author

☑ If the check is contributed by a third party (i.e., not by the developers of Check_MK), the name and email address of the contributor should be added as a comment, right after the header.

Coding style: Readability, looks and indents.

☑ Avoid long lines. Ideally, your lines shouldn't exceed 100 chars.

☑ Use four spaces to indent your code. Don't use tab chars! And if you really can't live without tabs, set the tab width to 8 spaces.

Coding style: File Header

☑ For checks which are supposed to be part of the official Check_MK project the file header with the copyright information must be present. This will be automatically created if you call 'make headers' in the main source directory.

Coding style: Example agent output

Including example output of the agent is very helpful for understanding how the check parser works.

Coding style: Example agent output (cont)

☑ TCP-Agent based checks **must** include an output example of the agent. If the agent output can have different formats or output styles, then put an example for each kind of style the check supports (e.g.: the output of multipath -l has changed its layout between SLES 10 and SLES 11).

☑ For SNMP based checks, at least include examples if the kind of output is remarkable in some respect.

Coding style: Use of lambda functions

When it comes to `parse_function`, `inventory_function` and `check_function`, the usage of `lambda` functions is only allowed in order to reuse *existing* functions while providing some additional argument. Example:

```
"inventory_function" : inventory_foobar_generic(info, "temperature")
```

It is not allowed to implement the function itself as `lambda` expression. Example:

```
# This is bad, ugly and unreadable code!!
```

`'check_function' : lambda _no_item, _no_params, info: `

```
(0, "Memory used: %s" % get_bytes_human_readable(int(info[0][0]))),
```

Manpages

☑ Each check must have a check man page. This should be:

- complete
- precise
- terse
- helpful!

☑ Information that must be contained in the check description:

- What does the check exactly do?
- A definition under which circumstances the check status will change to WARN/CRIT?

Manpages (cont)

Which devices are supported by the check?

Does the check require some configuration of the agent or some separate agent plugin? (example: the logwatch check requires the agent plugin mk_logwatch to be installed)

Service Descriptions

☑ Checks doing the same should always have the same (consistent) service description. Examples:

CPU utilization services must be named CPU utilization.

Temperature services must begin with the word Temperature.

Services for main RAM usage should be named Memory used.

Services for fans should be named Fan or Fan %s.

Services for power supplies should be named Power Supply or Power Supply %s.

☑ Service descriptions should be capitalized like English titles, e.g. "Source of Output"

Forbidden Things

⚠ Never use a global import statement in a check file

⚠ Do not use datetime for date/time parsing. Use time. It can do all you need, really !!!

⚠ Do not use any other modules, except: sys, os, time, socket

⚠ If you need regular expressions, use the function regex(). Do not use re directly.

⚠ Neither the check function nor the inventory function may use the print command, or otherwise output any data to stdout or stderr, or communicate with the outside world in any other way. An rare exception to this are checks which need a dedicated data storage (such as logwatch: it keeps unread log messages in files).

Forbidden Things (cont)

⚠ Never fetch SNMP data that is not actually used in the check or inventory function.

Temperature checks

☑ The item name should reflect the kind of temperature being monitored. Please refer to the following table to make sure that the same kinds of temperatures get the same item.

Ambient: Built-in sensor measuring ambient air temperature

External: An external, freely placeable sensor connected to the device

System: System mainboard temperature

CPU: CPU temperature

☑ To ensure that all temperature checks work in the same way, use the check_temperature function in temperature.include.

☑ The check group should be temperature.

☑ check_temperature can handle device levels and status in various ways configurable in the temperature WATO rule. Do not pass both device status and device levels to check_temperature - if a device provides levels, pass those and not the status.

☑ Some devices can output temperature in various units, and specify which unit it is. In those cases, pass the temperature in the unit the device states, along with the unit as the dev_unit parameter to check_temperature.

☑ Some devices have a very large number of similar temperature sensors, where one item per sensor would be unreasonable. (Dozens of ambient temperature sensors in a small device do not really provide more information than a single one.) In those cases, use the check_temperature_list function defined in temperature.include. Use the temperature check group just as you would for regular temperature checks.

Setting default values for configuration variables

☑ Default values for check parameters (e.g. `switch_cpu_default_levels`) must be chosen in a way that they make sense for *everybody*, not just for your special case. If case you are unsure, rather choose too loose than too tight levels. This helps avoid false alarms.

☑ If you set default values, add a short comment about how you came to choose said values. If it is merely a rough estimate, document that it is, if you got them from a very specific source, document where you got them.

Reuse of configuration variables

☑ If the same configuration variable is used in multiple checks, it must be set to a default value in **all** checks and the values must be identical!

Error handling

☑ Your check should assume that the agent is always producing valid data. It should **not** try to handle cases when the agent output is broken. Reason: broken agent output is already handled by Check_MK via Python exceptions. Intercepting these exceptions in your check code makes debugging of broken outputs much more difficult.

☑ Do not handle cases in the agent output for which you have no indication that they can actually happen.

int() vs. saveint() and float

☑ vs. `savefloat()` `int()` will throw an exception if the argument is not a valid number string (or if it is empty). Check_MK will catch the exception and make the check result "UNKNOWN" with an appropriate error message. `saveint()`, however, will assume 0 if the argument cannot be converted to a valid integer.

By **dlicudi**
cheatography.com/dlicudi/

Published 11th August, 2015.
Last updated 11th August, 2015.

int() vs. saveint() and float (cont)

☑ Use `saveint()` in all cases when you know or suspect that your device may supply invalid data, **but** the check should work with the rest of the data and produce useful results. Disadvantage: you may never find out that the device has supplied invalid data, because the check wont tell you !

☑ Use `int()` in all other cases, e.g. if you want to be notified with an exception if the check has received invalid data from your device. In most cases this is what you want !

Interpretation of levels

☑ Many checks have parameters defining warning and critical levels which are compared to an actual value. Please observe the following important rules and conventions if you are writting such checks.

☑ Warning and critical levels should always be checked with >= and <=. Example: a check monitors the length of a mail queue. The critical upper level is at 100. This means that if the length is exactly 100, the check should already be critical. There might be a few exceptions to this where this wouldn't make sense.

☑ If there are both upper and lower levels, the labelling should be: Warning at or above ___, Critical at or above ___, Warning at or below ___ and Critical at or below ___.

☑ If there are both upper and lower levels, the labelling should be: Warning at or above ___, Critical at or above ___, Warning at or below ___ and Critical at or below ___.

return versus yield

☑ A check function producing several subresults (e.g. current usage and growth) must use the `yield` function for returning these results. On the other hand, check generating exactly one result must use `return`.

check_info[...] keys

☑ Do not add keys here which are not used. The only mandatory keys are "`service_description`" and "`check_function`". Add "`has_perfdata`" and other keys with a boolean value only if its value is `True`.

Various

Here are some frequent errors and further mixed guidelines:

☑ If your check is accompanied by an agent plugin, you should observe the following rules:

Put it into share/check_mk/agents for UNIX like systems and make it executable (mode 755).

Put it into share/check_mk/agents/windows for Windows.

Do not add a file extension like .sh or .py.

For shell scripts, add #!/bin/sh in the first line. Use #!/bin/bash only if the BASH is really required.

Add the standard Check_MK file header with the GPL notice.

Make sure that the plugin does not do any harm even if installed on a system where the check in question is not relevant or does not work.

Make sure that the check manpage tells the user that the plugin is needed and which additional software needs to be installed in order to make it work.

The plugin must not output a section header if the tool or technology to be monitored does not exist on the system.

If the plugin needs a configuration file, expect it in $MK_CONFDIR and give it the same name as the plugin, but with the extension .cfg, and with any mk_ prefix removed.

Various (cont)

☑ A check which does not get the information which is needed decide whether or not the check is OK, must simply return None. This can be the case when a check with an item can not found the data matching this item in the agent output or SNMP data. Another possible situation is when the data provided by the agent or SNMP is completely empty.

When a check returns None, Check_MK will produce an UNKNOWN state with a state output which tells the user that this thing could not be found.

☑ The state markers (!) and (!!) must only be used in checks which can go warning or critical for several different reasons, like sub-checks.

☑ Your check must also work with Nagios as Core. If you use functions or variables from *.include files then you must declare them in check_info in the key "includes" and you must then test our check with Nagios as the core.

By **dlicudi**
cheatography.com/dlicudi/

Published 11th August, 2015.
Last updated 11th August, 2015.