Pycinga Documentation

Release 1.0.0

Hurricane Labs

Mar 05, 2018

Contents

CHAPTER 1

Overview

Pycinga is a Python library for writing Icinga plug-ins. Writing an Icinga plug-in typically involves conforming to a large list of guidelines, and this library removes the boilerplate necessary in writing your own plug-ins.

tutorial Start here for a quick start guide.

API Documentation The complete API documentation, organized by module.

CHAPTER 2

Issues

If any issues are found with the library, please file an issue with the Pycinga issue tracker. Tickets are triaged quickly and you should receive a prompt response.

CHAPTER 3

Contributing

Pycinga is an open source project which welcomes contributions which can be anything from simple documentation fixes to new features. To contribute, fork the project on GitHub and send a pull request.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

4.1 API Documentation

Pycinga contains one main top-level package, *pycinga*.

4.1.1 pycinga – Python Library for Writing Icinga Plugins

This package provides all the modules for writing an Icinga plugin with Python. The package file itself exports the constants used throughout the library.

pycinga.**version = '1.0.0'**
Current version of Pycinga

pycinga.**PerfData**
Alias for *pycinga.perf_data.PerfData*

pycinga.**Plugin**
Alias for *pycinga.plugin.Plugin*

pycinga.**Range**
Alias for *pycinga.range.Range*

pycinga.**Response**
Alias for *pycinga.response.Response*

pycinga.**Status**
Alias for *pycinga.status.Status*

pycinga.**OK**
A *Status* object representing the OK response status.

pycinga.**WARNING**
A *Status* object representing the WARNING response status.

pycinga.**CRITICAL**
A *Status* object representing the CRITICAL response status.

pycinga.**UNKNOWN**
A *Status* object representing the UNKNOWN response status.

Sub-modules:

perf_data – Performance Data Classes

Tools for creating performance data for Icinga plugin responses. If you're adding performance data to a *Response* object, then *set_perf_data()* can be called instead of having to create an entire *PerfData* object.

class pycinga.perf_data.**PerfData**(*label*, *value*[, *uom=None*[, *warn=None*[, *crit=None*[, *minval=None*[, *maxval=None*]]]]])
Creates a new object representing a single performance data item for an Icinga response.

Performance data is extra key/value data that can be returned along with a response. The performance data is not used immediately by Icinga itself, but can be extracted by 3rd party tools and can often be helpful additional information for system administrators to view. The *label* can be any string, but *value* must be a numeric value.

Raises `ValueError` if any of the parameters are invalid. The exact nature of the error is in the human readable message attribute of the exception.

Parameters

- *label*: Label for the performance data. This must be a string.
- *value*: Value of the data point. This must be a number whose characters are in the class of *[-0-9.]*
- *uom* (optional): Unit of measure. This must only be *%*, *s* for seconds, *c* for continous data, or a unit of bit space measurement ("b", "kb", etc.)
- *warn* (optional): Warning range for this metric.
- *crit* (optional): Critical range for this metric.
- *minval* (optional): Minimum value possible for this metric, if one exists.
- *maxval* (optional): Maximum value possible for this metric, if one exists.

value
The value of this metric.

warn
The warning range of this metric. This return value of this will always be a *Range* object, even if it was set with a string.

crit
The critical range of this metric. This return value of this will always be a *Range* object, even if it was set with a string.

minval
The minimum value possible for this metric. This doesn't make a lot of sense if the *uom* is "%", since that is obviously going to be 0, but this will return whatever was set.

maxval
The maximum value possible for this metric. This doesn't make a lot of sense if the *uom* is "%", since that is obviously going to be 100, but this will return whatever was set.

uom
The unit of measure (UOM) for this metric.

__str__()
Returns the proper string format that should be outputted in the plugin response string. This format is documented in depth in the Icinga developer guidelines, but in general looks like this:

'label'=value[UOM];[warn];[crit];[min];[max]

`plugin` – Plugin Class

This module provides the Plugin class, which is the basic class which encapsulates a single plugin. This is the class which should be subclassed when creating new plugins.

class pycinga.plugin.**Plugin**([*argv=sys.argv*])
Instantiates a plugin, setting up the options and arguments state. Initialization by itself shouldn't do much, since the plugin should run when *check()* is called.

This init method will parse the arguments given in `args` and will set the results on the `options` attribute. If no `args` are given, the command line arguments given to the whole Python application will be used.

All plugins parse standard command line arguments that are required by the Icinga developer guidelines:

- `hostname` - Set via `-H` or `--hostname`, this should be the host that this check targets, if applicable.
- `warning` - Set via `-w` or `--warning`, this should be a valid range in which the value of the plugin is considered to be a warning.
- `critical` - Set via `-c` or `--critical`, this should be a valid range in which the value is considered to be critical.
- `timeout` - Set via `-t` or `--timeout`, this is an int value for the timeout of this check.
- `verbosity` - Set via `-v`, where additional `v` means more verbosity. Example: `-vvv` will set `options.verbosity` to 3.

Subclasses can define additional options by creating `Action` instances and assigning them to class attributes. The easiest way to make an `Action` is to use Python's built-in `argparse` methods. The following is an example plugin which adds a simple string argument::

```
class MyPlugin(Plugin):
    parser = ArgumentParser()
    parser.add_argument("--your-name", dest="your_name", type="string")
```

Instantiating the above plugin will result in the value of the new argument being available in `options.your_name`.

options
Dictionary of parsed command line options and their values. As an example, to get the `hostname` passed in via the command line::

```
options.hostname
```

args
Array of additional positional arguments passed in via the command line. For example, if you call the plugin with `./plugin 1 2 3`, then `options.args` will return `[1,2,3]`.

check()
This method is what should be called to run this plugin and return a proper *Response* object. Subclasses are expected to implement this.

response_for_value(*value*, *message=None*)
This method is meant to be used by plugin implementers to return a valid *Response* object for the given value. The status of this response is determined based on the warning and critical ranges given via the command line, which the plugin automatically parses.

An optional `message` argument may be provided to set the message for the Response object. Note that this can easily be added later as well by simply setting the message attribute on the response object returned.

Creating a response using this method from *check()* makes it trivial to calculate the value, grab a response, set some performance metrics, and return it.

response - Response Class

Contains the class which represents a response for Icinga. This encapsulates the response format that Icinga expects.

class pycinga.response.**Response**([*status=None*[, *message=None*]])
Icinga responses are expected to be in a very specific format, and this class allows these responses to easily be built up and extracted in the proper format.

This class makes it easy to set the status, message, and performance data for a response.

Parameters

- *status* (optional): A *Status* object representing the status of the response.
- *message* (optional): An information message to include with the output.

set_perf_data(*label*, *value*, *uom=None*, *warn=None*, *crit=None*, *minval=None*, *maxval=None*)
Adds performance data to the response. Performance data is shown in the Icinga GUI and can be used by 3rd party programs to build graphs or other informational output. There are many options to this method. They are the same as the initialization parameters for a *PerfData* object.

See also:

PerfData

exit()
This prints out the response to `stdout` and exits with the proper exit code.

__str__()
The string format of this object is the valid Icinga output format. The response format is expected to be the following:

```
status: information|performance data
```

An example of realistic output:

```
OK: 27 users logged in|users=27;0:40;0:60;0;
```

range - Tools for Working with Icinga Ranges

Contains a class to represent a range that adheres to the range format defined by Icinga.

class pycinga.range.**Range**(*value*)

Initializes an Icinga range with the given value. The value should be in the Icinga range format, which is the following:

```
[@]start:end
```

Notes:

- `start` must be less than or equal to `end`
- Ranges by default are exclusive. A range of `10:20` will match values that are `< 10 OR >20`.
- `@` means the range is *inclusive*. So `@10:20` is valid in the case that the value is `>= 10 AND <= 20`.
- If `start` or `end` is `~`, this value is negative or positive inifinity, respectively. A range of `~:20` will match values that are `> 20` only.
- If `start` is not given, then it is assumed to be 0.
- If `end` is not given, but a `:` exists, then `end` is assumed to be infinity. Example: `5:` would match `< 5`.

in_range(*value*)

Tests whether `value` is in this range.

__str__()

Turns this range object back into a valid range string which can be passed to another plugin or used for debug output. The string returned from here should generally be equivalent to the value given to the constructor, but sometimes it can be slightly different. However, it will always be functionally equivalent.

Examples:

```
>> str(Range("@10:20")) == "@10:20"
>> str(Range("10")) == "10"
>> str(Range("10:")) == "10:~"
```

class pycinga.range.**RangeValueError**

This exception is raised when an invalid value is passed to *Range*. The message of this exception will contain a human readable explanation of the error.

`status` - Icinga Status Objects

This module provides the Status class, which encapsulates a status code for Icinga.

class pycinga.status.**Status**(*name*, *exit_code*)

Creates a new status object for Icinga with the given name and exit code.

Note: In general, this should never be called since the standard statuses are exported from `pycinga`.

Python Module Index

p

Index