

---

# **systemd-logging Documentation**

***Release 1.0.1***

**Igor ‘idle sign’ Starikov**

**Jan 28, 2022**



---

## Contents

---

<b>1</b>	<b>Description</b>	<b>3</b>
<b>2</b>	<b>Requirements</b>	<b>5</b>
<b>3</b>	<b>Table of Contents</b>	<b>7</b>
3.1	Quickstart . . . . .	7
3.2	API . . . . .	8
	<b>Python Module Index</b>	<b>11</b>
	<b>Index</b>	<b>13</b>



<https://github.com/idlesign/systemd-logging>



# CHAPTER 1

---

## Description

---

*Simplifies logging for systemd*

- No need to compile (pure Python), uses `libsystemd.so`.
- Simplified configuration.
- Just logging. Nothing more.





## CHAPTER 2

---

### Requirements

---

1. Python 3.6+



### 3.1 Quickstart

```
import logging

from systemdlogging.toolbox import init_systemd_logging

# This one line in most cases would be enough.
# By default it attaches systemd logging handler to a root Python logger.
init_systemd_logging() # Returns True if initialization went fine.

# Now you can use logging as usual.
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

logger.debug('My debug message')

try:
    raise ValueError('Log me please')
except ValueError:
    # Additional context can be passed in extra.context.
    logger.exception('Something terrible just happened', extra={
        'message_id': True, # Generate message ID automatically.
        'context': {
            'FIELD1': 'one',
            'FIELD2': 'two',
        }
    }, stack_info=True)
```

## 3.2 API

**class** `systemdlogging.toolbox.SystemdFormatter` (*fmt=None, datefmt=None, style='%'*)

Formatter for systemd.

Initialize the formatter with specified format strings.

Initialize the formatter either with the specified format string, or a default as described above. Allow for specialized date formatting with the optional `datefmt` argument. If `datefmt` is omitted, you get an ISO8601-like (or RFC 3339-like) format.

Use a style parameter of `'%'`, `'{'` or `'$'` to specify that you want to use one of `%`-formatting, `str.format()` (`{}`) formatting or `string.Template` formatting in your format string.

Changed in version 3.2: Added the `style` parameter.

**format** (*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using `LogRecord.getMessage()`. If the formatting string uses the time (as determined by a call to `usesTime()`), `formatTime()` is called to format the event time. If there is exception information, it is formatted using `formatException()` and appended to the message.

**class** `systemdlogging.toolbox.SystemdHandler` (*level=0*)

Allows passing log messages to systemd.

Additional information may be passed in `extra` dictionary:

- `context` - Dictionary with additional information to attach to a message.
- **message\_id** - Message ID (usually UUID to assign to the message). If `True` ID will be generated automatically

Fills in the following fields automatically:

- `CODE_FILE` - Filename
- `CODE_LINE` - Code line number
- `CODE_FUNC` - Code function name
- `CODE_MODULE` - Python module name
- `LOGGER` - Logger name
- `THREAD_ID` - Thread ID if any
- `THREAD_NAME` - Thread name if any
- `PROCESS_NAME` - Process name if any
- `PRIORITY` - Priority based on logging level

**Optionally fills in:**

- `TRACEBACK` - Traceback information if available
- `STACK` - Stacktrace information if available
- `MESSAGE_ID` - Message ID (if `context.message_id=True`)

Initializes the instance - basically setting the formatter to `None` and the filter list to empty.

**emit** (*record: logging.LogRecord*)

Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a `NotImplementedError`.

**syslog\_id** = ''

SYSLOG\_IDENTIFIER to add to message. If not set command name is used.

`systemdlogging.toolbox.check_for_systemd()` → bool

Checks whether systemd library is available and the current process is run under systemd (and thus its output is connected to journald).

`systemdlogging.toolbox.init_systemd_logging(*, logger: Optional[logging.Logger] = None, syslog_id: str = '')` → bool

Initializes logging to send log messages to systemd.

Returns boolean indicating initialization went fine (see also `check_for_systemd()`).

If it wasn't one may either to fallback to another logging handler or leave it as it is possibly sacrificing some log context.

#### Parameters

- **logger** – Logger to attach systemd logging handler to. If not set handler is attached to a root logger.
- **syslog\_id** – Value to be used in SYSLOG\_IDENTIFIER message field.

`systemdlogging.toolbox.log_message(*, level: int, msg: str, context: Optional[dict] = None)` → bool

Sends a message to systemd.

**Warning:** Low level function. You may want to use more convenient `init_systemd_logging()` or `SystemdHandler` and `SystemdFormatter`.

**Natively supported fields:** <http://0pointer.de/public/systemd-man/systemd.journal-fields.html>

#### Parameters

- **level** – Logging level.
- **msg** – Message text to send.
- **context** – Additional context to send within message.



### S

`systemdlogging.toolbox`, [8](#)





## C

`check_for_systemd()` (in module `systemdlogging.toolbox`), 9

## E

`emit()` (`systemdlogging.toolbox.SystemdHandler` method), 8

## F

`format()` (`systemdlogging.toolbox.SystemdFormatter` method), 8

## I

`init_systemd_logging()` (in module `systemdlogging.toolbox`), 9

## L

`log_message()` (in module `systemdlogging.toolbox`), 9

## S

`syslog_id` (`systemdlogging.toolbox.SystemdHandler` attribute), 9

`SystemdFormatter` (class in `systemdlogging.toolbox`), 8

`SystemdHandler` (class in `systemdlogging.toolbox`), 8

`systemdlogging.toolbox` (module), 8