uwsgiconf Documentation

Release 1.0.0

Igor ‘idle sign‘ Starikov

Apr 13, 2023

Contents

1 Description **3**

2 Requirements **5**

3 Table of Contents **7**

3.1 Quickstart . . . 7
3.2 Presets . . . 9
3.3 Command-line interface (CLI) . . . 12
3.4 Python uwsgi module stub . . . 14
3.5 Contrib . . . 29
3.6 FAQ . . . 31
3.7 Configuration [Dynamic/Runtime] . . . 32
3.8 Configuration [Static] . . . 50

4 Get involved into uwsgiconf **151**

Python Module Index **153**

Index **155**

https://github.com/idlesign/uwsgiconf

CHAPTER 1

Description

Configure uWSGI from your Python code

If you think you known uWSGI you're probably wrong. It is always more than you think it is. There are so many subsystems and options (800+) it is difficult to even try to wrap your mind around.

uwsgiconf allowing to define uWSGI configurations in Python tries to improve things the following ways:

- It structures options for various subsystems using classes and methods;
- It uses docstrings and sane naming to facilitate navigation;
- It ships some useful presets to reduce boilerplate code;
- It encourages configuration reuse;
- It comes with CLI to facilitate configuration;
- It features easy to use and documented **uwsgi stub** Python module;
- It offers **runtime** package, similar to **uwsgidecorators**, but with more abstractions;
- It features integration with Django Framework;
- It is able to generate configuration files for Systemd, Upstart.
- It can use `pyuwsgi`.

Consider using IDE with autocompletion and docstings support to be more productive with uwsgiconf.

By that time you already know that **uwsgiconf** is just another configuration method. Why?

CHAPTER 2

Requirements

1. Python 3.7+
2. `click` package (optional, for CLI)
3. uWSGI (`uwsgi` or `pyuwsgi`)

CHAPTER 3

Table of Contents

3.1 Quickstart

3.1.1 Install

You can get and install **uwsgiconf** from PyPI using `pip`:

```
$ pip install uwsgiconf
```

CLI

uwsgiconf expects `click` package available for CLI but won't install this dependency by default.

Use the following command to install **uwsgiconf** with `click`:

```
$ pip install uwsgiconf[cli]
```

3.1.2 Using a preset to run Python web application

Let's make `uwsgicfg.py`. There we configure uWSGI using nice `PythonSection` preset to run our web app.

```
from uwsgiconf.config import configure_uwsgi
from uwsgiconf.presets.nice import PythonSection


def get_configurations():
    """This should return one or more Section or Configuration objects.
    In such a way you can configure more than one uWSGI instance in the same place.

    """
    my_app_dir = '/home/idle/myapp/'
```

(continues on next page)

(continued from previous page)

```
    section = PythonSection(
        # Reload uWSGI when this file is updated.
        touch_reload=__file__,

        params_python=dict(
            # Let's add something into Python path.
            search_path='/opt/apps_shared/',
        ),

        wsgi_module=my_app_dir + 'wsgi.py',

        # We'll redirect logs into a file.
        log_into=my_app_dir + 'app.log',

        # If your uWSGI has no basic plugins embedded
        # (i.e. not from PyPI) you can give uwsgiconf a hint:
        # embedded_plugins=False,

    ).networking.register_socket(
        # Make app available at http://127.0.0.1:8000
        PythonSection.networking.sockets.http('127.0.0.1:8000'))

    return section


# Almost done. One more thing:
configure_uwsgi(get_configurations)
```

Now we are ready to use this configuration:

```
$ uwsgiconf compile > myconf.ini
$ uwsgi myconf.ini

; or instead just
$ uwsgiconf run
```

3.1.3 Configuration with multiple sections

Let's configure uWSGI to use Emperor Broodlord mode as described here using `Broodlord` preset.

```
from uwsgiconf.config import Section, Configuration
from uwsgiconf.presets.empire import Broodlord

emperor, zerg = Broodlord(

    zerg_socket='/tmp/broodlord.sock',
    zerg_count=40,
    zerg_die_on_idle=30,

    vassals_home='/etc/vassals',
    vassal_queue_items_sos=10,

    # We'll use the same basic params both for Broodlord Emperor and his zergs.
    section_emperor=(Section().
```

(continues on next page)

(continued from previous page)

```
        # NOTE. Here we use a shortcut for ``set_basic_params`` methods:
        # E.g.: instead of `master_process.set_basic_params(enable=True)`
        # you say `master_process(enable=True)`.
        # But in that case you won't get any arg hints from you IDE.
        master_process(enable=True).
        workers(count=1).
        logging(no_requests=True).
        python.set_wsgi_params(module='werkzeug.testapp:test_app')
    ),

).configure()

# Bind Emperor to socket.
emperor.networking.register_socket(Section.networking.sockets.default(':3031'))

# Put Emperor and zerg sections into configuration.
multisection_config = Configuration([emperor, zerg])
```

3.2 Presets

Presets are means to reduce boilerplate code.

Use them as shortcuts to spend less time on configuring.

3.2.1 Preset: Empire

This preset offers configuration helpers related to Emperor and his vassals.

class uwsgiconf.presets.empire.**Broodlord**(*zerg_socket: str*, ***, *zerg_die_on_idle: int = None*, *vassals_home: Union[str, List[str]] = None*, *zerg_count: int = None*, *vassal_overload_sos_interval: int = None*, *vassal_queue_items_sos: int = None*, *section_emperor: uwsgiconf.config.Section = None*, *section_zerg: uwsgiconf.config.Section = None*)

This mode is a way for a vassal to ask for reinforcements to the Emperor.

Reinforcements are new vassals spawned on demand generally bound on the same socket.

> **Warning:** If you are looking for a way to dynamically adapt the number of workers of an instance, check the Cheaper subsystem - adaptive process spawning mode.
>
> *Broodlord mode is for spawning totally new instances.*

- http://uwsgi-docs.readthedocs.io/en/latest/Broodlord.html

Parameters

- **zerg_socket** – Unix socket to bind server to.
- **zerg_die_on_idle** – A number of seconds after which an idle zerg will be destroyed.
- **vassals_home** – Set vassals home.

- **zerg_count** – Maximum number of zergs to spawn.
- **vassal_overload_sos_interval** – Ask emperor for reinforcement when overloaded. Accepts the number of seconds to wait between asking for a new reinforcements.
- **vassal_queue_items_sos** – Ask emperor for sos if backlog queue has more items than the value specified
- **section_emperor** – Custom section object.
- **section_zerg** – Custom section object.

configure() → Tuple[uwsgiconf.config.Section, uwsgiconf.config.Section]
Configures broodlord mode and returns emperor and zerg sections.

3.2.2 Preset: Nice

This preset offers nice configuration defaults.

class uwsgiconf.presets.nice.**Section**(*name: str = None*, ***, *touch_reload: Union[str, List[str]] = None*, *workers: int = None*, *threads: Union[int, bool] = None*, *mules: int = None*, *owner: str = None*, *log_into: str = None*, *log_dedicated: bool = None*, *process_prefix: str = None*, *ignore_write_errors: bool = None*, ***kwargs*)
Basic nice configuration.

Parameters

- **name** – Section name.
- **touch_reload** – Reload uWSGI if the specified file or directory is modified/touched.
- **workers** – Spawn the specified number of workers (processes). Default: workers number equals to CPU count.
- **threads** – Number of threads per worker or `True` to enable user-made threads support.
- **mules** – Number of mules to spawn.
- **owner** – Set process owner user and group.
- **log_into** – Filepath or UDP address to send logs into.
- **log_dedicated** – If `True` all logging will be handled with a separate thread in master process.
- **process_prefix** – Add prefix to process names.
- **ignore_write_errors** – If `True` no annoying SIGPIPE/write/writev errors will be logged, and no related exceptions will be raised.

  Note: Usually such errors could be seen on client connection cancelling and are safe to ignore.

- **kwargs** –

get_log_format_default() → str
Returns default log message format.

Note: Some params may be missing.

classmethod get_bundled_static_path(*filename: str*) → str
Returns a full path to a static HTML page file bundled with uwsgiconf.

Parameters **filename** – File name to return path to.

Examples:

- 403.html
- 404.html
- 500.html
- 503.html

configure_maintenance_mode(*trigger: Union[str, pathlib.Path], response: str*)
Allows maintenance mode when a certain response is given for every request if a trigger is set.

Parameters

- **trigger** – This triggers maintenance mode responses. Should be a path to a file: if file exists, maintenance mode is on.
- **response** – Response to to give in maintenance mode.

  Supported:

  1. File path - this file will be served in response.
  2. URLs starting with `http` - requests will be redirected there using 302.

     This is often discouraged, because it may have search ranking implications.

  3. Prefix `app` will replace your entire app with a maintenance one.

     Using this also prevents background tasks registration and execution (including scheduler, timers, signals).

     – If the value is *app* - the default maintenance application bundled with uwsgiconf would be used.

     – Format `app::<your-module>:<your-app-function>` instructs uwsgiconf to load your function as a maintenance app. E.g.: app::my_pack.my_module:my_func

configure_owner(*owner: str = 'www-data'*)
Shortcut to set process owner data.

Parameters **owner** – Sets user and group. Default: `www-data`.

configure_https_redirect()
Enables HTTP to HTTPS redirect.

configure_certbot_https(*domain: str*, *webroot: str*, ***, *address: str = None*, *allow_shared_sockets: bool = None*, *http_redirect: bool = False*)
Enables HTTPS using certificates from Certbot https://certbot.eff.org.

Note: This relies on `webroot` mechanism of Certbot - https://certbot.eff.org/docs/using.html#webroot

Sample command to get a certificate: `certbot certonly --webroot -w /webroot/path/ -d mydomain.org`

Parameters

- **domain** – Domain name certificates issued for (the same as in `-d` option in the above command).
- **webroot** – Directory to store challenge files to get and renew the certificate (the same as in `-w` option in the above command).
- **address** – Address to bind socket to.
- **allow_shared_sockets** – Allows using shared sockets to bind to privileged ports. If not provided automatic mode is enabled: shared are allowed if current user is not root.
- **http_redirect** – Redirect HTTP requests to HTTPS if certificates exist.

configure_logging_json()
Configures uWSGI output to be json-formatted.

class uwsgiconf.presets.nice.**PythonSection**(*name: str = None*, ***, *params_python: dict = None*, *wsgi_module: str = None*, *wsgi_callable: Union[str, Callable] = None*, *embedded_plugins: Optional[bool] = True*, *require_app: bool = True*, *threads: Union[bool, int] = True*, ***kwargs*)

Basic nice configuration using Python plugin.

Parameters

- **name** – Section name.
- **params_python** – See Python plugin basic params.
- **wsgi_module** – WSGI application module path or filepath.

  Example: mypackage.my_wsgi_module – read from *application* attr of mypackage/my_wsgi_module.py mypackage.my_wsgi_module:my_app – read from *my_app* attr of mypackage/my_wsgi_module.py
- **wsgi_callable** – WSGI application callable name. Default: application.
- **embedded_plugins** – This indicates whether plugins were embedded into uWSGI, which is the case if you have uWSGI from PyPI.
- **require_app** – Exit if no app can be loaded.
- **threads** – Number of threads per worker or `True` to enable user-made threads support.
- **kwargs** –

3.3 Command-line interface (CLI)

uwsgiconf comes with CLI (`click` package required) to facilitate configuration.

```
; To install uwsgiconf with click:
$ pip install uwsgiconf[cli]
```

3.3.1 Compile

Compiles classic uWSGI configuration from a given *uwsgiconf* configuration module (or from the default one - `uwsgicfg.py`).

Note: Be sure that your configuration module defines `configuration` attribute. It mush hold one or more `Configuration` or `Section` (those will be automatically casted to configurations) objects. Callable as attribute value is supported.

```
; This compiles uwsgicfg.py from the current directory
; into .ini and prints that out:
$ uwsgiconf compile

; This compiles there/thisfile.py file:
$ uwsgiconf compile there/thisfile.py

; Add "> target_file.ini" to redirect output (configuration) into a file.
```

3.3.2 Run

Runs uWSGI using configuration from a given *uwsgiconf* configuration module (or from the default one - `uwsgicfg.py`).

Note: uWSGI process will replace `uwsgiconf` process.

```
; This runs uWSGI using uwsgicfg.py from the current directory.
$ uwsgiconf run

; This runs uWSGI using configuration from there/thisfile.py:
$ uwsgiconf run there/thisfile.py
```

3.3.3 Probe plugins

Shows available uWSGI plugins.

```
$ uwsgiconf probe_plugins
```

3.3.4 Systemd and other configs

You can generate configuration files to launch `uwsgiconf` automatically using system facilities.

Config contents in sent to stdout and could be redirected into a file.

```
$ uwsgiconf sysinit systemd
$ uwsgiconf sysinit upstart
```

Usage example for Systemd:

```
; Generate and save config into `myapp.service` file
$ uwsgiconf sysinit --project myapp > myapp.service

; Copy config into standard location
$ sudo cp myapp.service /etc/systemd/system/

; Reload available configs information and run service
$ sudo sh -c "systemctl daemon-reload; systemctl start myapp.service"

; Watch application log realtime (if syslog is used)
$ journalctl -fu myapp.service
```

3.4 Python uwsgi module stub

uwsgiconf comes with documented **uwsgi** module that you can import instead of `import uwsgi`.

```
# Instead of
import uwsgi

# you can do.
from uwsgiconf import uwsgi
```

That way **uwsgi** will be available runtime as usual, besides you will get autocompletion and hints in IDE, and won't get `ImportError` when run without **uwsgi**.

This also will facilitate your testing a bit, for those simple cases when you won't expect any result from **uwsgi** function.

> **Warning:** This is a stub module, so it doesn't really implement functions defined in it. Use it for documentation purposes.

`uwsgiconf.uwsgi_stub.`**`is_stub = True`**
Indicates whether stub is used instead of real *uwsgi* module.

`uwsgiconf.uwsgi_stub.`**`SPOOL_IGNORE = 0`**
Spooler function result.

Ignore this task, if multiple languages are loaded in the instance all of them will fight for managing the task. This return values allows you to skip a task in specific languages.

- http://uwsgi-docs.readthedocs.io/en/latest/Spooler.html#setting-the-spooler-function-callable

`uwsgiconf.uwsgi_stub.`**`SPOOL_OK = -2`**
Spooler function result.

The task has been completed, the spool file will be removed.

- http://uwsgi-docs.readthedocs.io/en/latest/Spooler.html#setting-the-spooler-function-callable

`uwsgiconf.uwsgi_stub.`**`SPOOL_RETRY = -1`**
Spooler function result.

Something is temporarily wrong, the task will be retried at the next spooler iteration.

- http://uwsgi-docs.readthedocs.io/en/latest/Spooler.html#setting-the-spooler-function-callable

`uwsgiconf.uwsgi_stub.`**`SymbolsImporter = None`**
SymbolsImporter type.

uwsgiconf.uwsgi_stub.**SymbolsZipImporter = None**
SymbolsZipImporter type.

uwsgiconf.uwsgi_stub.**ZipImporter = None**
ZipImporter type.

uwsgiconf.uwsgi_stub.**applications = None**
Applications dictionary mapping mountpoints to application callables.

Note: Can be `None`.

- http://uwsgi.readthedocs.io/en/latest/Python.html#application-dictionary

uwsgiconf.uwsgi_stub.**buffer_size = 0**
The current configured buffer size in bytes.

uwsgiconf.uwsgi_stub.**cores = 0**
Detected number of processor cores.

uwsgiconf.uwsgi_stub.**env = {}**
Request environment dictionary.

uwsgiconf.uwsgi_stub.**has_threads = False**
Flag indicating whether thread support is enabled.

uwsgiconf.uwsgi_stub.**hostname = b''**
Current host name.

uwsgiconf.uwsgi_stub.**magic_table = {}**
Current mapping of configuration file "magic" variables.
- http://uwsgi.readthedocs.io/en/latest/Configuration.html#magic-variables

uwsgiconf.uwsgi_stub.**numproc = 0**
Number of workers (processes) currently running.

uwsgiconf.uwsgi_stub.**opt = {}**
The current configuration options, including any custom placeholders.

uwsgiconf.uwsgi_stub.**post_fork_hook = <bound method _PostForkHooks.run of <class 'uwsgicon**
Function to be called after process fork (spawning a new worker/mule).

uwsgiconf.uwsgi_stub.**spooler = <bound method Spooler._process_message_raw of <class 'uwsgi**
Function to be called for spooler messages processing.

uwsgiconf.uwsgi_stub.**sockets = []**
Current list of file descriptors for registered sockets.

uwsgiconf.uwsgi_stub.**start_response = None**
Callable spitting UWSGI response.

uwsgiconf.uwsgi_stub.**started_on = 0**
uWSGI's startup Unix timestamp.

uwsgiconf.uwsgi_stub.**unbit = False**
Unbit internal flag.

uwsgiconf.uwsgi_stub.**version = b'0.0.0'**
The uWSGI version string.

uwsgiconf.uwsgi_stub.**version_info = (0, 0, 0, 0, b'')**
Five-elements version number tuple.

uwsgiconf.uwsgi_stub.**mule_msg_hook**(*message: bytes*)
Registers a function to be called for each mule message.

uwsgiconf.uwsgi_stub.**accepting**()
Called to notify the master that the worker is accepting requests, this is required for `touch_chain_reload` to work.

uwsgiconf.uwsgi_stub.**add_cron**(*signal: int*, *minute: int*, *hour: int*, *day: int*, *month: int*, *weekday: int*) → bool
Adds cron. The interface to the uWSGI signal cron facility. The syntax is

---

Note: The last 5 arguments work similarly to a standard crontab, but instead of "*", use -1, and instead of "/2", "/3", etc. use -2 and -3, etc.

Parameters

- **signal** – Signal to raise.
- **minute** – Minute 0-59. Defaults to *each*.
- **hour** – Hour 0-23. Defaults to *each*.
- **day** – Day of the month number 1-31. Defaults to *each*.
- **month** – Month number 1-12. Defaults to *each*.
- **weekday** – Day of a the week number. Defaults to *each*. 0 - Sunday 1 - Monday 2 - Tuesday 3 - Wednesday 4 - Thursday 5 - Friday 6 - Saturday

Raises **ValueError** – If unable to add cron rule.

uwsgiconf.uwsgi_stub.**add_file_monitor**(*signal: int*, *filename: str*)
Maps a specific file/directory modification event to a signal.

Parameters

- **signal** – Signal to raise.
- **filename** – File or a directory to watch for its modification.

Raises **ValueError** – If unable to register monitor.

uwsgiconf.uwsgi_stub.**add_ms_timer**(*signal: int*, *period: int*)
Add a millisecond resolution timer.

Parameters

- **signal** – Signal to raise.
- **period** – The interval (milliseconds) at which to raise the signal.

Raises **ValueError** – If unable to add timer.

uwsgiconf.uwsgi_stub.**add_rb_timer**(*signal: int*, *period: int*, *repeat: int = 0*)
Add a red-black timer.

Parameters

- **signal** – Signal to raise.
- **period** – The interval (seconds) at which the signal is raised.
- **repeat** – How many times to send signal. Will stop after ther number is reached. Default: 0 - infinitely.

Raises **ValueError** – If unable to add timer.

uwsgiconf.uwsgi_stub.**add_timer**(*signal: int*, *period: int*)
Add timer.

Parameters

- **signal** – Signal to raise.
- **period** – The interval (seconds) at which to raise the signal.

Raises **ValueError** – If unable to add timer.

uwsgiconf.uwsgi_stub.**add_var**(*name: str*, *value: str*) → bool
Registers custom request variable.

Can be used for better integration with the internal routing subsystem.

Parameters

- **name** –
- **value** –

Raises **ValueError** – If buffer size is not enough.

uwsgiconf.uwsgi_stub.**alarm**(*name: str*, *message: str*)
Issues the given alarm with the given message.

Note: to register an alarm use `section.alarms.register_alarm(section.alarms.alarm_types.log('myalarm'))`

Parameters

- **name** –
- **message** – Message to pass to alarm.

uwsgiconf.uwsgi_stub.**async_connect**(*socket: str*) → int
Issues socket connection. And returns a file descriptor or -1.

- http://uwsgi.readthedocs.io/en/latest/Async.html

Parameters **socket** –

uwsgiconf.uwsgi_stub.**async_sleep**(*seconds: int*) → bytes
Suspends handling the current request and passes control to the next async core.

- http://uwsgi.readthedocs.io/en/latest/Async.html

Parameters **seconds** – Sleep time, in seconds.

uwsgiconf.uwsgi_stub.**cache_clear**(*cache: str*)
Clears cache with the given name.

Parameters **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**cache_dec**(*key: str*, *value: int = 1*, *expires: int = None*, *cache: str = None*) → bool
Decrements the specified key value by the specified value.

- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.9.html#math-for-cache

Parameters

- **key** –
- **value** –
- **expires** – Expire timeout (seconds).
- **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**cache_del**(*key: str*, *cache: str = None*) → bool
Deletes the given cached key from the cache.

Parameters

- **key** – The cache key to delete.
- **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**cache_div**(*key: str*, *value: int = 2*, *expires: int = None*, *cache: str = None*) → bool
Divides the specified key value by the specified value.

- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.9.html#math-for-cache

Parameters

- **key** –
- **value** –
- **expires** – Expire timeout (seconds).
- **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**cache_exists**(*key: str*, *cache: str = None*) → bool
Checks whether there is a value in the cache associated with the given key.

Parameters

- **key** – The cache key to check.
- **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**cache_get**(*key: str*, *cache: str = None*) → Optional[bytes]
Gets a value from the cache.

Parameters

- **key** – The cache key to get value for.
- **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**cache_inc**(*key: str*, *value: int = 1*, *expires: int = None*, *cache: str = None*) → bool
Increments the specified key value by the specified value.

- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.9.html#math-for-cache

Parameters

- **key** –
- **value** –
- **expires** – Expire timeout (seconds).
- **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**cache_keys**(*cache: str = None*) → List[T]
Returns a list of keys available in cache.

Parameters **cache** (*str*) – Cache name with optional address (if @-syntax is used).

Raises **ValueError** – If cache is unavailable.

uwsgiconf.uwsgi_stub.**cache_mul**(*key: str*, *value: int = 2*, *expires: int = None*, *cache: str = None*) → bool
Multiplies the specified key value by the specified value.

- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.9.html#math-for-cache

Parameters

- **key** –
- **value** –
- **expires** – Expire timeout (seconds).
- **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**cache_num**(*key: str*, *cache: str = None*) → Optional[int]

Gets the 64bit number from the specified item.

- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.9.html#math-for-cache

Parameters

- **key** – The cache key to get value for.
- **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**cache_set**(*key: str*, *value: str*, *expires: int = None*, *cache: str = None*) → bool

Sets the specified key value.

Parameters

- **key** –
- **value** –
- **expires** – Expire timeout (seconds).
- **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**cache_update**(*key: str*, *value: str*, *expires: int = None*, *cache: str = None*) → bool

Updates the specified key value.

Parameters

- **key** –
- **value** –
- **expires** – Expire timeout (seconds).
- **cache** – Cache name with optional address (if @-syntax is used).

uwsgiconf.uwsgi_stub.**call**(*func_name: bytes*, **args*) → bytes

Performs an [RPC] function call with the given arguments.

Parameters

- **func_name** – Function name to call with optional address (if @-syntax is used).
- **args** –

uwsgiconf.uwsgi_stub.**chunked_read**(*timeout: int*) → bytes

Reads chunked input.

- http://uwsgi.readthedocs.io/en/latest/Chunked.html
- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.13.html#chunked-input-api

Parameters **timeout** – Wait timeout (seconds).

Raises **IOError** – If unable to receive chunked part.

uwsgiconf.uwsgi_stub.**chunked_read_nb**() → bytes

Reads chunked input without blocking.

- http://uwsgi.readthedocs.io/en/latest/Chunked.html
- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.13.html#chunked-input-api

Raises **IOError** – If unable to receive chunked part.

uwsgiconf.uwsgi_stub.**cl**() → int
Returns current post content length.

uwsgiconf.uwsgi_stub.**close**(*fd: int*)
Closes the given file descriptor.
Parameters **fd** – File descriptor.

uwsgiconf.uwsgi_stub.**connect**(*socket: str*, *timeout: int = 0*) → int
Connects to the socket.
Parameters
- **socket** – Socket name.
- **timeout** – Timeout (seconds).

uwsgiconf.uwsgi_stub.**connection_fd**() → int
Returns current request file descriptor.

uwsgiconf.uwsgi_stub.**disconnect**()
Drops current connection.

uwsgiconf.uwsgi_stub.**embedded_data**(*symbol_name: str*) → bytes
Reads a symbol from the uWSGI binary image.
- http://uwsgi.readthedocs.io/en/latest/Embed.html

Parameters **symbol_name** – The symbol name to extract.

Raises **ValueError** – If symbol is unavailable.

uwsgiconf.uwsgi_stub.**extract**(*fname: str*) → bytes
Extracts file contents.
Parameters **fname** –

uwsgiconf.uwsgi_stub.**farm_get_msg**() → Optional[bytes]
Reads a mule farm message.
- http://uwsgi.readthedocs.io/en/latest/Embed.html

Raises **ValueError** – If not in a mule

uwsgiconf.uwsgi_stub.**farm_msg**(*farm: str, message: Union[str, bytes]*)
Sends a message to the given farm.
Parameters
- **farm** – Farm name to send message to.
- **message** –

uwsgiconf.uwsgi_stub.**get_logvar**(*name: str*) → bytes
Return user-defined log variable contents.
- http://uwsgi.readthedocs.io/en/latest/LogFormat.html#user-defined-logvars

Parameters **name** –

uwsgiconf.uwsgi_stub.**green_schedule**() → bool
Switches to another green thread.

Note: Alias for `suspend`.

- http://uwsgi.readthedocs.io/en/latest/Async.html#suspend-resume

uwsgiconf.uwsgi_stub.**i_am_the_spooler**() → bool
Returns flag indicating whether you are the Spooler.

uwsgiconf.uwsgi_stub.**in_farm**(*name: str*) → Optional[bool]
Returns flag indicating whether you (mule) belong to the given farm. Returns None is not in a mule.
Parameters **name** – Farm name.

uwsgiconf.uwsgi_stub.**is_a_reload**() → bool
Returns flag indicating whether reloading mechanics is used.

uwsgiconf.uwsgi_stub.**is_connected**(*fd: int*) → bool
Checks the given file descriptor.
Parameters **fd** – File descriptor

uwsgiconf.uwsgi_stub.**is_locked**(*lock_num: int = 0*) → bool
Checks for the given lock.

Note: Lock 0 is always available.

Parameters **lock_num** – Lock number.

Raises **ValueError** – For Spooler or invalid lock number

uwsgiconf.uwsgi_stub.**listen_queue**(*socket_num: int = 0*) → int
Returns listen queue (backlog size) of the given socket.
Parameters **socket_num** – Socket number.

Raises **ValueError** – If socket is not found

uwsgiconf.uwsgi_stub.**lock**(*lock_num: int = 0*)
Sets the given lock.

Note: Lock 0 is always available.

Parameters **lock_num** – Lock number.

Raises **ValueError** – For Spooler or invalid lock number

uwsgiconf.uwsgi_stub.**log**(*message: str*) → bool
Logs a message.
Parameters **message** –

uwsgiconf.uwsgi_stub.**log_this_request**()
Instructs uWSGI to log current request data.

uwsgiconf.uwsgi_stub.**logsize**() → int
Returns current log size.

uwsgiconf.uwsgi_stub.**loop**() → Optional[str]
Returns current event loop name or None if loop is not set.

uwsgiconf.uwsgi_stub.**masterpid**() → int
Return the process identifier (PID) of the uWSGI master process.

uwsgiconf.uwsgi_stub.**mem**() → Tuple[int, int]
Returns memory usage tuple of ints: (rss, vsz).

uwsgiconf.uwsgi_stub.**metric_dec**(*key: str*, *value: int = 1*) → bool
Decrements the specified metric key value by the specified value.
Parameters

- **key** –

- **value** –

uwsgiconf.uwsgi_stub.**metric_div**(*key: str*, *value: int = 1*) → bool
Divides the specified metric key value by the specified value.

Parameters

- **key** –
- **value** –

uwsgiconf.uwsgi_stub.**metric_get**(*key: str*) → int
Returns metric value by key.

Parameters **key** –

uwsgiconf.uwsgi_stub.**metric_inc**(*key: str*, *value: int = 1*) → bool
Increments the specified metric key value by the specified value.

Parameters

- **key** –
- **value** –

uwsgiconf.uwsgi_stub.**metric_mul**(*key: str*, *value: int = 1*) → bool
Multiplies the specified metric key value by the specified value.

Parameters

- **key** –
- **value** –

uwsgiconf.uwsgi_stub.**metric_set**(*key: str*, *value: int*) → bool
Sets metric value.

Parameters

- **key** –
- **value** –

uwsgiconf.uwsgi_stub.**metric_set_max**(*key: str*, *value: int*) → bool
Sets metric value if it is greater that the current one.

Parameters

- **key** –
- **value** –

uwsgiconf.uwsgi_stub.**metric_set_min**(*key: str*, *value: int*) → bool
Sets metric value if it is less that the current one.

Parameters

- **key** –
- **value** –

uwsgiconf.uwsgi_stub.**micros**() → int
Returns uWSGI clock microseconds.

uwsgiconf.uwsgi_stub.**mule_get_msg**(*signals: bool = None*, *farms: bool = None*, *buffer_size: int = 65536*, *timeout: int = -1*) → bytes
Block until a mule message is received and return it.

This can be called from multiple threads in the same programmed mule.

Parameters

- **signals** – Whether to manage signals.

- **farms** – Whether to manage farms.
- **buffer_size** –
- **timeout** – Seconds.

Raises **ValueError** – If not in a mule.

uwsgiconf.uwsgi_stub.**mule_id**() → int
Returns current mule ID. 0 if not a mule (e.g. worker).

uwsgiconf.uwsgi_stub.**mule_msg**(*message: Union[str, bytes], mule_farm: Union[str, int] = None*) → bool
Sends a message to a mule(s)/farm.

Parameters

- **message** –
- **mule_farm** – Mule ID, or farm name.

Raises **ValueError** – If no mules, or mule ID or farm name is not recognized.

uwsgiconf.uwsgi_stub.**offload**(*filename: str*) → bytes
Offloads a file.

> **Warning:** Currently not implemented.

Parameters **filename** –

Raises **ValueError** – If unable to offload.

uwsgiconf.uwsgi_stub.**parsefile**(*fpath: str*)
Parses the given file.

Currently implemented only Spooler file parsing.

Parameters **fpath** –

uwsgiconf.uwsgi_stub.**ready**() → bool
Returns flag indicating whether we are ready to handle requests.

uwsgiconf.uwsgi_stub.**ready_fd**() → bool
Returns flag indicating whether file description related to request is ready.

uwsgiconf.uwsgi_stub.**recv**(*fd: int, maxsize: int = 4096*) → bytes
Reads data from the given file descriptor.

Parameters

- **fd** –
- **maxsize** – Chunk size (bytes).

uwsgiconf.uwsgi_stub.**register_rpc**(*name: str, func: Callable*) → bool
Registers RPC function.

- http://uwsgi.readthedocs.io/en/latest/RPC.html

Parameters

- **name** –
- **func** –

Raises **ValueError** – If unable to register function

uwsgiconf.uwsgi_stub.**register_signal**(*number: int, target: str, func: Callable*)
Registers a signal handler.

Parameters

- **number** – Signal number.
- **target** –
  - `workers` - run the signal handler on all the workers
  - `workerN` - run the signal handler only on worker N
  - `worker/worker0` - run the signal handler on the first available worker
  - `active-workers` - run the signal handlers on all the active [non-cheaped] workers
  - `mules` - run the signal handler on all of the mules
  - `muleN` - run the signal handler on mule N
  - `mule/mule0` - run the signal handler on the first available mule
  - `spooler` - run the signal on the first available spooler
  - `farmN/farm_XXX` - run the signal handler in the mule farm N or named XXX
  - http://uwsgi.readthedocs.io/en/latest/Signals.html#signals-targets
- **func** –

Raises **ValueError** – If unable to register

uwsgiconf.uwsgi_stub.**reload**() → bool
Gracefully reloads uWSGI.
- http://uwsgi.readthedocs.io/en/latest/Management.html#reloading-the-server

uwsgiconf.uwsgi_stub.**request_id**() → int
Returns current request number (handled by worker on core).

uwsgiconf.uwsgi_stub.**route**(*name: str*, *args_str: str*) → int
Registers a named route for internal routing subsystem.

Parameters

- **name** – Route name
- **args_str** – Comma-separated arguments string.

uwsgiconf.uwsgi_stub.**rpc**(*address: Optional[bytes]*, *func_name: bytes*, **args*) → bytes
Performs an RPC function call with the given arguments.
- http://uwsgi.readthedocs.io/en/latest/RPC.html

Parameters

- **address** –
- **func_name** – Function name to call.
- **args** –

Raises **ValueError** – If unable to call RPC function.

uwsgiconf.uwsgi_stub.**rpc_list**() → Tuple[bytes, ...]
Returns registered RPC functions names.

uwsgiconf.uwsgi_stub.**send**(*fd_or_data: Union[int, bytes]*, *data: bytes = None*) → bool
Puts data into file descriptor.
- One argument. Data to write into request file descriptor.
- Two arguments. 1. File descriptor; 2. Data to write.

Parameters

- **fd_or_data** –
- **data** –

uwsgiconf.uwsgi_stub.**sendfile**(*fd_or_name: Union[int, str], chunk_size: int = 0, start_pos: int = 0, filesize: int = 0*) → Optional[bool]

Runs a sendfile.

Parameters

- **fd_or_name** – File path or descriptor number.
- **chunk_size** – Not used.
- **start_pos** –
- **filesize** – Filesize. If `0` will be determined automatically.

uwsgiconf.uwsgi_stub.**send_to_spooler**(*message: Dict[bytes, bytes] = None*, ***kwargs*)

Send data to the The uWSGI Spooler. Also known as spool().

> **Warning:** Either *message* argument should contain a dictionary this message dictionary will be constructed from *kwargs*.

Parameters

- **message** – The message to spool. Keys and values are bytes.
- **kwargs** – Possible kwargs (these are also reserved *message* argument dictionary keys):
  - **spooler: The spooler (id or directory) to use.** Specify the ABSOLUTE path of the spooler that has to manage this task
  - priority: Number. The priority of the message. Larger - less important.

    > **Warning:** This works only if you enable *order_tasks* option in *spooler.set_basic_params()*.

    This will be the subdirectory in the spooler directory in which the task will be placed, you can use that trick to give a good-enough prioritization to tasks.

    Note: This is for systems with few resources. For better approach use multiple spoolers.

  - **at: Unix time at which the task must be executed.** The task will not be run until the 'at' time is passed.
  - **body: A binary body to add to the message,** in addition to the message dictionary itself. Use this key for objects bigger than 64k, the blob will be appended to the serialized uwsgi packet and passed back t o the spooler function as the 'body' argument.

uwsgiconf.uwsgi_stub.**set_logvar**(*name: str*, *value: str*)

Sets log variable.

Parameters

- **name** –
- **value** –

uwsgiconf.uwsgi_stub.**set_spooler_frequency**(*seconds: int*) → bool
Sets how often the spooler runs.
Parameters **seconds** –

uwsgiconf.uwsgi_stub.**set_user_harakiri**(*timeout: int = 0*)
Sets user level harakiri.
Parameters **timeout** – Seconds. 0 disable timer.

uwsgiconf.uwsgi_stub.**set_warning_message**(*message: str*) → bool
Sets a warning. This will be reported by pingers.
Parameters **message** –

uwsgiconf.uwsgi_stub.**setprocname**(*name: str*)
Sets current process name.
Parameters **name** –

uwsgiconf.uwsgi_stub.**signal**(*num: int*, *remote: str = ''*)
Sends the signal to master or remote.
Parameters

- **num** – Signal number.
- **remote** – Remote address.

Raises

- **ValueError** – If remote rejected the signal.
- **IOError** – If unable to deliver to remote.

uwsgiconf.uwsgi_stub.**signal_received**() → int
Get the number of the last signal received.

Used in conjunction with signal_wait.
- http://uwsgi-docs.readthedocs.io/en/latest/Signals.html#signal-wait-and-signal-received

uwsgiconf.uwsgi_stub.**signal_registered**(*num: int*) → Optional[int]
Verifies the given signal has been registered.
Parameters **num** –

uwsgiconf.uwsgi_stub.**signal_wait**(*num: int = None*) → str
Waits for the given of any signal.

Block the process/thread/async core until a signal is received. Use signal_received to get the number of the signal received. If a registered handler handles a signal, signal_wait will be interrupted and the actual handler will handle the signal.
- http://uwsgi-docs.readthedocs.io/en/latest/Signals.html#signal-wait-and-signal-received

Parameters **num** (*int*) –

Raises **SystemError** – If something went wrong.

uwsgiconf.uwsgi_stub.**spool**(*message: Dict[bytes, bytes] = None*, ***kwargs*)
Send data to the The uWSGI Spooler. Also known as spool().

> **Warning:** Either *message* argument should contain a dictionary this message dictionary will be constructed from *kwargs*.

Parameters

- **message** – The message to spool. Keys and values are bytes.
- **kwargs** – Possible kwargs (these are also reserved *message* argument dictionary keys):
  - **spooler: The spooler (id or directory) to use.** Specify the ABSOLUTE path of the spooler that has to manage this task
  - priority: Number. The priority of the message. Larger - less important.

    Warning: This works only if you enable *order_tasks* option in *spooler.set_basic_params()*.

    This will be the subdirectory in the spooler directory in which the task will be placed, you can use that trick to give a good-enough prioritization to tasks.

    Note: This is for systems with few resources. For better approach use multiple spoolers.

  - **at: Unix time at which the task must be executed.** The task will not be run until the 'at' time is passed.
  - **body: A binary body to add to the message,** in addition to the message dictionary itself. Use this key for objects bigger than 64k, the blob will be appended to the serialized uwsgi packet and passed back t o the spooler function as the 'body' argument.

uwsgiconf.uwsgi_stub.**spooler_get_task**(*path: str*) → Optional[dict]
Returns a spooler task information.
Parameters **path** – The relative or absolute path to the task to read.

uwsgiconf.uwsgi_stub.**spooler_jobs**() → List[str]
Returns a list of spooler jobs (filenames in spooler directory).

uwsgiconf.uwsgi_stub.**spooler_pid**() → int
Returns first spooler process ID

uwsgiconf.uwsgi_stub.**spooler_pids**() → List[int]
Returns a list of all spooler processes IDs.

uwsgiconf.uwsgi_stub.**stop**() → Optional[bool]
Stops uWSGI.

uwsgiconf.uwsgi_stub.**suspend**() → bool
Suspends handling of current coroutine/green thread and passes control to the next async core.
- http://uwsgi.readthedocs.io/en/latest/Async.html#suspend-resume

uwsgiconf.uwsgi_stub.**total_requests**() → int
Returns the total number of requests managed so far by the pool of uWSGI workers.

uwsgiconf.uwsgi_stub.**unlock**(*lock_num: int = 0*)
Unlocks the given lock.

Note: Lock 0 is always available.

Parameters **lock_num** – Lock number.

Raises **ValueError** – For Spooler or invalid lock number

uwsgiconf.uwsgi_stub.**wait_fd_read**(*fd: int*, *timeout: int = None*) → bytes
Suspends handling of the current request until there is something to be read on file descriptor.

May be called several times before yielding/suspending to add more file descriptors to the set to be watched.
- http://uwsgi-docs.readthedocs.io/en/latest/Async.html#waiting-for-i-o

Parameters

- **fd** – File descriptor number.
- **timeout** – Timeout. Default: infinite.

Raises **OSError** – If unable to read.

uwsgiconf.uwsgi_stub.**wait_fd_write**(*fd: int*, *timeout: int = None*) → bytes
Suspends handling of the current request until there is nothing more to be written on file descriptor.

May be called several times to add more file descriptors to the set to be watched.
- http://uwsgi-docs.readthedocs.io/en/latest/Async.html#waiting-for-i-o

Parameters

- **fd** – File descriptor number.
- **timeout** – Timeout. Default: infinite.

Raises **OSError** – If unable to read.

uwsgiconf.uwsgi_stub.**websocket_handshake**(*security_key: str = None*, *origin: str = None*, *proto: str = None*)
Waits for websocket handshake.

Parameters

- **security_key** – Websocket security key to use.
- **origin** – Override `Sec-WebSocket-Origin`.
- **proto** – Override `Sec-WebSocket-Protocol`.

Raises **IOError** – If unable to complete handshake.

uwsgiconf.uwsgi_stub.**websocket_recv**(*request_context=None*) → bytes
Receives data from websocket.

Parameters **request_context** –

Raises **IOError** – If unable to receive a message.

uwsgiconf.uwsgi_stub.**websocket_recv_nb**(*request_context=None*) → bytes
Receives data from websocket (non-blocking variant).

Parameters **request_context** –

Raises **IOError** – If unable to receive a message.

uwsgiconf.uwsgi_stub.**websocket_send**(*message: str*, *request_context=None*)
Sends a message to websocket.

Parameters

- **message** – data to send
- **request_context** –

Note: uWSGI 2.1+

Raises **IOError** – If unable to send a message.

uwsgiconf.uwsgi_stub.**websocket_send_binary**(*message: str*, *request_context=None*)
Sends binary message to websocket.

Parameters

- **message** – data to send
- **request_context** –

Note: uWSGI 2.1+

Raises **IOError** – If unable to send a message.

uwsgiconf.uwsgi_stub.**worker_id**() → int
Returns current worker ID. 0 if not a worker (e.g. mule).

uwsgiconf.uwsgi_stub.**workers**() → Tuple[dict, ...]
Gets statistics for all the workers for the current server.

Returns tuple of dicts.

uwsgiconf.uwsgi_stub.**i_am_the_lord**(*legion_name: str*) → bool
Returns flag indicating whether you are the lord of the given legion.

- http://uwsgi.readthedocs.io/en/latest/Legion.html#legion-api

Parameters **legion_name** –

uwsgiconf.uwsgi_stub.**lord_scroll**(*legion_name: str*) → bool
Returns a Lord scroll for the Legion.

- http://uwsgi.readthedocs.io/en/latest/Legion.html#lord-scroll-coming-soon

Parameters **legion_name** –

uwsgiconf.uwsgi_stub.**scrolls**(*legion_name: str*) → List[str]
Returns a list of Legion scrolls defined on cluster.

Parameters **legion_name** –

3.5 Contrib

Additional integrations with third parties.

3.5.1 Django uwsigify

`uwsgify` adds integration with Django Framework.

First add `uwsgify` into `INSTALLED_APPS`.

```
INSTALLED_APPS = [
    ...
    'uwsgiconf.contrib.django.uwsgify',
    ...
]
```

uwsgi_run

`uwsgi_run` management command runs uWSGI to serve your Django-based project.

```
$ ./manage.py uwsgi_run

; Options are available, use --help switch to get help.
$ ./manage.py uwsgi_run --help
```

Now your project is up and running on `http://127.0.0.1:8000`.

By default the command runs your project using some defaults, but you can configure it to your needs with the help of `uwsgicfg.py` (constructed in a usual for **uwsgiconf** manner) placed near your `manage.py`.

```
from uwsgiconf.config import configure_uwsgi


def get_configurations():

    from os.path import dirname, abspath, join
    from uwsgiconf.presets.nice import PythonSection


    section = PythonSection.bootstrap(
        'http://127.0.0.1:8000',
        wsgi_module=join(dirname(abspath(__file__)), 'wsgi.py')
    )

    ...

    return section


configure_uwsgi(get_configurations)
```

Note: **Embedding.** if you're using **pyuwsgi** having uWSGI and your entire project compiled into a single binary, and your *manage.py* is the entrypoint, use **–embedded** option: `myproject uwsgi_run --embedded`.

uwsgi_reload

`uwsgi_reload` management command reloads uWSGI master process, workers.

```
$ ./manage.py uwsgi_reload

; Options are available, use --help switch to get help.
$ ./manage.py uwsgi_reload --help
```

uwsgi_stop

`uwsgi_stop` management command allows you to shutdown uWSGI instance.

```
$ ./manage.py uwsgi_stop
```

(continues on next page)

(continued from previous page)

```
; Options are available, use --help switch to get help.
$ ./manage.py uwsgi_stop --help
```

uwsgi_stats

`uwsgi_stats` management command allows you to dump uWSGI configuration and current stats into the log.

```
$ ./manage.py uwsgi_stats
```

uwsgi_log

`uwsgi_log` management command allows you to manage uWSGI log related stuff.

```
$ ./manage.py uwsgi_log --rotate

; Options are available, use --help switch to get help.
$ ./manage.py uwsgi_log --help
```

uwsgi_sysinit

`uwsgi_sysinit` management command allows you to generate system service configs (e.g. `systemd`) to start your Django project on system start.

```
; Dump config to file.
$ ./manage.py uwsgi_sysinit > myapp.service

; Wire up the service config into system directory and start service
$ sudo systemctl enable --now myapp.service

; Watch application log realtime
$ sudo journalctl -fu myapp.service
```

3.6 FAQ

3.6.1 How to get through

There are some many options, how to start?

Start with a preset configuration. For example, if you have a Python web-app try out `uwsgiconf.presets.nice.PythonSection`. Basic things to do are: define `wsgi_module` and do `.networking.register_socket`.

This should already give a more or less decent configuration.

After that you can skim through option groups (such as `.networking`, `.main_process`, `.workers` etc.) and deep into uWSGI abilities.

3.6.2 Use from virtualenv

I have a virtualenv in venv/ directory (where I have uWSGI and uwsgiconf) and configuration module outside of it, how do I run it?

You can try the following trick (from directory containing `venv/` and `uwsgicfg.py`):

```
$ venv/bin/uwsgiconf run
```

3.6.3 Unknown config directive

I use `PythonSection` *for configuration and get* **[strict-mode] unknown config directive: wsgi-file** *on start. What's that?*

uwsgiconf enables configuration options check (aka `strict-mode`) by default.

If uWSGI plugin which provides some options is not available, you'll get the message. That's because `PythonSection` by default won't instruct uWSGI to load Python plugin (since if you get uWSGI from PyPI you already have Python and a bunch of other plugins embedded, so there's no need to load them).

If you get that message most probably uWSGI is provided by your OS distribution (e.g. on Debian you'll need to install plugin packages separately from uWSGI itself).

In that case you can try to set `embedded_plugins=False` for `PythonSection` (see Quickstart example).

Another option is to quickly fire up `uWSGI` to check what plugins are embedded (the same can be achieved with `$ uwsgiconf probe-plugins` command).

uwsgiconf can also do it for you automatically on configuration stage:

```
Section(embedded_plugins=Section.embedded_plugins_presets.PROBE)
```

Using the above, `embedded_plugins` will be inhabited by plugins actually available in uWSGI.

3.7 Configuration [Dynamic/Runtime]

uwsgiconf comes with `runtime` package which is similar to **uwsgidecorators** but offers different abstractions to provide useful shortcuts and defaults.

Various modules from that package can be imported and used runtime to configure different aspects of **uWSGI**, such as *caching*, *locks*, *signals*, *spooler*, *rpc*, etc.

3.7.1 Alarms

3.7.2 Asynced

3.7.3 Caching

```
from uwsgiconf.runtime.caching import Cache

# We'll access preconfigured cache named `mycache`.
cache = Cache('mycache')

```

(continues on next page)

(continued from previous page)

```
key_exists = 'mykey' in cache

def my_setter(key):
    if key == 'anotherkey':
        return 'yes'
    return 'no'

# Getting cached value and populating it if required in one pass:
yes_or_no = cache.get('anotherkey', setter=my_setter)
```

class uwsgiconf.runtime.caching.**Cache**(*name: str*, *, *timeout: int = None*)

Interface for uWSGI Caching subsystem.

> **Warning:** To use this helper one needs to configure cache(s) in uWSGI config beforehand.
>
> E.g.: `section.caching.add_cache('mycache', 100)`

Parameters

- **name** – Cache name with optional address (if @-syntax is used).
- **timeout** – Expire timeout (seconds). Default: 300 (5 minutes). Use 0 to not to set a timeout (not to expire).

  Note: This value is ignore if cache is configured not to expire.

keys

Returns a list of keys available in cache.

Raises **ValueError** – If cache is unavailable.

clear()

Clears cache the cache.

get(*key: str*, *, *default: Any = None*, *as_int: bool = False*, *setter: Callable = None*) → Union[str, int]

Gets a value from the cache.

Parameters

- **key** – The cache key to get value for.
- **default** – Value to return if none found in cache.
- **as_int** – Return 64bit number instead of str.
- **setter** – Setter callable to automatically set cache value if not already cached. Required to accept a key and return a value that will be cached.

set(*key: str*, *value: Any*, *, *timeout: int = None*) → bool

Sets the specified key value.

Parameters

- **key** –
- **value** –

Note: This value will be casted to string as uWSGI cache works with strings.

- **timeout** – 0 to not to expire. Object default is used if not set.

delete(*key: str*)
Deletes the given cached key from the cache.

Parameters **key** – The cache key to delete.

incr(*key: str*, *, *delta: int = 1*) → bool
Increments the specified key value by the specified value.

Parameters

- **key** –
- **delta** –

decr(*key: str*, *, *delta: int = 1*) → bool
Decrements the specified key value by the specified value.

Parameters

- **key** –
- **delta** –

mul(*key: str*, *, *value: int = 2*) → bool
Multiplies the specified key value by the specified value.

Parameters

- **key** –
- **value** –

div(*key: str*, *, *value: int = 2*) → bool
Divides the specified key value by the specified value.

Parameters

- **key** –
- **value** –

3.7.4 Control

```
from uwsgiconf.runtime.control import harakiri_imposed, reload

@harakiri_imposed(1)
def doomed():
    """Master process will kill this function after 1 sec."""

# or

with harakiri_imposed(30):
    # Master will kill worker if code under that manager won't finish in 30 sec.

# We'll reload uWSGI.
reload()
```

class uwsgiconf.runtime.control.**harakiri_imposed**(*, *timeout: int*)

Decorator and context manager.

Allows temporarily setting harakiri timeout for a function or a code block.

Note: This is for workers, mules and spoolers.

Examples:

```
@harakiri_imposed(1)
def doomed():
    do()
```

```
with harakiri_imposed(10):
    do()
```

Parameters **timeout** – Timeout (seconds) before harakiri.

3.7.5 Locking

```
from uwsgiconf.runtime.locking import lock

@lock()
def locked():
    """This function will be locked with default (0) lock."""
    ...

 # or

with lock(2):
    # Code under this context manager will be locked with lock 2.
    ...
```

class uwsgiconf.runtime.locking.**Lock**(*num: int = 0*)

Locks related stuff.

Lock number 0 is always available. More locks need to be registered with `.config.locking.set_basic_params(count=X)` where `X` is the number of locks.

Note: The same lock should be released before next acquiring.

Can be used as context manager:

```
with Lock():
    do()
```

Can de used as a decorator:

```
@Lock()
def do():
    pass
```

Parameters **num** – Lock number (0-64). 0 is always available and is used as default.

is_set
"Checks whether the lock is active.

Raises **ValueError** – For Spooler or invalid lock number

acquire()
Sets the lock.

Raises **ValueError** – For Spooler or invalid lock number

release()
Unlocks the lock.

Raises **ValueError** – For Spooler or invalid lock number

uwsgiconf.runtime.locking.**lock**
Convenience alias for Lock.

alias of *uwsgiconf.runtime.locking.Lock*

3.7.6 Logging

uwsgiconf.runtime.logging.**variable_get**(*name: str*) → str
Return user-defined log variable contents.
- http://uwsgi.readthedocs.io/en/latest/LogFormat.html#user-defined-logvars

Parameters **name** –

3.7.7 Monitoring

uwsgiconf.runtime.monitoring.**register_file_monitor**(*filename: str*, *, *target: Union[str, int, uwsgiconf.runtime.signals.Signal] = None*)
Maps a specific file/directory modification event to a signal.
Parameters

- **filename** – File or a directory to watch for its modification.
- **target** – Existing signal to raise or Signal Target to register signal implicitly.

  Available targets:

  - workers - run the signal handler on all the workers
  - workerN - run the signal handler only on worker N
  - worker/worker0 - run the signal handler on the first available worker
  - active-workers - run the signal handlers on all the active [non-cheaped] workers
  - mules - run the signal handler on all of the mules
  - muleN - run the signal handler on mule N
  - mule/mule0 - run the signal handler on the first available mule
  - spooler - run the signal on the first available spooler
  - farmN/farm_XXX - run the signal handler in the mule farm N or named XXX

Raises **ValueError** – If unable to register monitor.

class uwsgiconf.runtime.monitoring.**Metric**(*name: str*)
User metric related stuff.

Note: One needs to register user metric beforehand. E.g.:: `section.monitoring.register_metric(section.monitoring.metric_types.absolute('mymetric'))`

Parameters **name** – Metric name.

value
Current metric value.

set(*value: int*, *, *mode: str = None*) → bool
Sets metric value.

Parameters

- **value** – New value.
- **mode** – Update mode.
  - None - Unconditional update.
  - max - Sets metric value if it is greater that the current one.
  - min - Sets metric value if it is less that the current one.

incr(*delta: int = 1*) → bool
Increments the specified metric key value by the specified value.

Parameters **delta** –

decr(*delta: int = 1*) → bool
Decrements the specified metric key value by the specified value.

Parameters **delta** –

mul(*value: int = 1*) → bool
Multiplies the specified metric key value by the specified value.

Parameters **value** –

div(*value: int = 1*) → bool
Divides the specified metric key value by the specified value.

Parameters **value** –

3.7.8 Mules

```
from uwsgiconf.runtime.mules import Mule, Farm

first_mule = Mule(1)

@first_mule.offload()
def for_mule(*args, **kwargs):
    # This function will be offloaded to and handled by mule 1.
    ...

farm_two = Farm('two')

@farm_two.offload()
```

(continues on next page)

(continued from previous page)

```
def for_farm(*args, **kwargs):
    # And this one will be offloaded to farm `two` and handled by any mule from that␣
↪farm.
    ...
```

uwsgiconf.runtime.mules.**mule_offload**(*mule_or_farm: Union[str, int, Mule, Farm] = None*) → Callable

Decorator. Use to offload function execution to a mule or a farm.

Parameters **mule_or_farm** – If not set, offloads to a first mule.

class uwsgiconf.runtime.mules.**Mule**(*id: int*)

Represents uWSGI Mule.

Note: Register mules before using this. E.g.: `section.workers.set_mules_params(mules=3)`

Parameters **id** – Mule ID. Enumeration starts with 1.

offload() → Callable

Decorator. Allows to offload function execution on this mule.

```
first_mule = Mule(1)

@first_mule.offload()
def for_mule(*args, **kwargs):
    # This function will be offloaded to and handled by mule 1.
    ...
```

classmethod **get_current_id**() → int

Returns current mule ID. Returns 0 if not a mule.

classmethod **get_current**() → Optional[uwsgiconf.runtime.mules.Mule]

Returns current mule object or None if not a mule.

classmethod **get_message**(**, signals: bool = True, farms: bool = False, buffer_size: int = 65536, timeout: int = -1*) → str

Block until a mule message is received and return it.

This can be called from multiple threads in the same programmed mule.

Parameters

- **signals** – Whether to manage signals.
- **farms** – Whether to manage farms.
- **buffer_size** –
- **timeout** – Seconds.

Raises **ValueError** – If not in a mule.

send(*message: Union[str, bytes]*) → bool

Sends a message to a mule(s)/farm.

Parameters **message** –

Raises **ValueError** – If no mules, or mule ID or farm name is not recognized.

class uwsgiconf.runtime.mules.**Farm**(*name: str, *, mules: List[int] = None*)

Represents uWSGI Mule Farm.

Note: Register farms before using this. E.g.: `section.workers.set_mules_params(farms=section.workers.mule_farm('myfarm', 2))`

Parameters

- **name** – Mule farm name.
- **mules** – Attached mules.

classmethod get_farms() → List[uwsgiconf.runtime.mules.Farm]
Returns a list of registered farm objects.

```
farms = Farm.get_farms()
first_farm = farms[0]
first_farm_first_mule = first_farm.mules[0]
```

offload() → Callable
Decorator. Allows to offload function execution on mules of this farm.

```
first_mule = Farm('myfarm')

@first_mule.offload()
def for_mule(*args, **kwargs):
    # This function will be offloaded to farm `myfarm` and handled by any␣
↪mule from that farm.
    ...
```

is_mine
Returns flag indicating whether the current mule belongs to this farm.

classmethod get_message() → str
Reads a mule farm message.

- http://uwsgi.readthedocs.io/en/latest/Embed.html

Raises **ValueError** – If not in a mule

send(*message: Union[str, bytes]*)
Sends a message to the given farm.

Parameters **message** –

3.7.9 Platform

Platform object is available in `uwsgi` module attribute:

```
from uwsgiconf.runtime.platform import uwsgi

rss, vsz = uwsgi.memory

print(uwsgi.config)

@uwsgi.postfork_hooks.add()
def db_close_connections():
    """This will be called after fork()."""
    print('Forked!')
```

class uwsgiconf.runtime.platform.**_Platform**

request
alias of *uwsgiconf.runtime.request._Request*

postfork_hooks
uWSGI is a preforking server, so you might need to execute a fixup tasks (hooks) after each fork(). Each hook will be executed in sequence on each process (worker/mule).

Note: The fork() happen before app loading, so there's no hooks for dynamic apps. But one can still move postfork hooks in a .py file and import it on server startup with *python.import_module()*.

---

alias of _PostForkHooks

workers_count = 0
Number of workers (processes) currently running.

cores_count = 0
Detected number of processor cores.

buffer_size = 0
The current configured buffer size in bytes.

threads_enabled = False
Flag indicating whether thread support is enabled.

started_on = 0
uWSGI's startup Unix timestamp.

apps_map = None
Applications dictionary mapping mountpoints to application callables.

hostname
Current host name.

config
The current configuration options, including any custom placeholders.

config_variables
Current mapping of configuration file "magic" variables.

worker_id
Returns current worker ID. 0 if not a worker (e.g. mule).

workers_info
Gets statistics for all the workers for the current server.

Returns tuple of dicts.

ready_for_requests
Returns flag indicating whether we are ready to handle requests.

master_pid
Return the process identifier (PID) of the uWSGI master process.

memory
Returns memory usage tuple of ints: (rss, vsz).

clock
Returns uWSGI clock microseconds.

get_listen_queue(*socket_num: int = 0*) → int
Returns listen queue (backlog size) of the given socket.

Parameters **socket_num** – Socket number.

Raises **ValueError** – If socket is not found

get_version(*, *as_tuple: bool = False*) → Union[str, Tuple[int, int, int, int, str]]
Returns uWSGI version string or tuple.

Parameters **as_tuple** –

class uwsgiconf.runtime.request.**_Request**
Current request information.

env
Request environment dictionary.

id
Returns current request number (handled by worker on core).

total_count
Returns the total number of requests managed so far by the pool of uWSGI workers.

fd
Returns current request file descriptor.

content_length
Returns current post content length.

log()
Instructs uWSGI to log current request data.

add_var(*name: str*, *value: str*) → bool
Registers custom request variable.

Can be used for better integration with the internal routing subsystem.

Parameters

- **name** –
- **value** –

Raises **ValueError** – If buffer size is not enough.

3.7.10 RPC

```
from uwsgiconf.runtime.rpc import register_rpc, make_rpc_call, get_rpc_list

@register_rpc()
def expose_me(arg1, arg2=15):
    print('RPC called %s' % arg1)

make_rpc_call('expose_me', ['value1'])

all_rpc = get_rpc_list()  # Registered RPC items list.
```

uwsgiconf.runtime.rpc.**register_rpc**(*name: str = None*) → Callable
Decorator. Allows registering a function for RPC.

- http://uwsgi.readthedocs.io/en/latest/RPC.html

```
@register_rpc()
def expose_me(arg1, arg2=15):
    print(f'RPC called {arg1}')
    return b'some'

make_rpc_call('expose_me', ['value1'])
```

Warning: Function expected to accept bytes args. Also expected to return bytes or `None`.

Parameters **name** – RPC function name to associate with decorated function.

uwsgiconf.runtime.rpc.**make_rpc_call**(*func_name: str*, ***, *args: Sequence[str] = None*, *remote: str = None*) → Optional[str]

Performs an RPC function call (local or remote) with the given arguments.

Parameters

- **func_name** – RPC function name to call.
- **args** (*Iterable*) – Function arguments.

  Warning: Strings are expected.

- **remote** –

Raises **ValueError** – If unable to call RPC function.

uwsgiconf.runtime.rpc.**get_rpc_list**() → List[str]

Returns registered RPC functions names.

3.7.11 Scheduling

```
from uwsgiconf.runtime.scheduling import register_timer_rb, register_cron

@register_timer_rb(10, repeat=2)
def repeat_twice():
    """This function will be called twice with 10 seconds interval
    (by default in in first available mule) using red-black tree based timer.

    """

@register_cron(day=-3, hour='10-18/2')
def do_something():
    """This will be run every 3rd day, from 10 till 18 every 2 hours."""
```

uwsgiconf.runtime.scheduling.**register_timer**(*period: int*, ***, *target: Union[str, int, uwsgiconf.runtime.signals.Signal] = None*) → Union[Callable, bool]

Add timer.

Can be used as a decorator:

```
@register_timer(3)
def repeat():
    do()
```

Parameters

- **period** (*int*) – The interval (seconds) at which to raise the signal.
- **target** – Existing signal to raise or Signal Target to register signal implicitly.

  Available targets:

  - `workers` - run the signal handler on all the workers
  - `workerN` - run the signal handler only on worker N
  - `worker`/`worker0` - run the signal handler on the first available worker
  - `active-workers` - run the signal handlers on all the active [non-cheaped] workers
  - `mules` - run the signal handler on all of the mules
  - `muleN` - run the signal handler on mule N
  - `mule`/`mule0` - run the signal handler on the first available mule
  - `spooler` - run the signal on the first available spooler
  - `farmN`/`farm_XXX` - run the signal handler in the mule farm N or named XXX

Raises **ValueError** – If unable to add timer.

uwsgiconf.runtime.scheduling.**register_timer_rb**(*period: int*, *, *repeat: int = None*, *target: Union[str, int, uwsgiconf.runtime.signals.Signal] = None*) → Union[Callable, bool]

Add a red-black timer (based on black-red tree).

```
@register_timer_rb(3)
def repeat():
    do()
```

Parameters

- **period** – The interval (seconds) at which the signal is raised.
- **repeat** – How many times to send signal. Will stop after ther number is reached. Default: None - infinitely.
- **target** – Existing signal to raise or Signal Target to register signal implicitly.

  Available targets:

  - `workers` - run the signal handler on all the workers
  - `workerN` - run the signal handler only on worker N
  - `worker`/`worker0` - run the signal handler on the first available worker
  - `active-workers` - run the signal handlers on all the active [non-cheaped] workers
  - `mules` - run the signal handler on all of the mules
  - `muleN` - run the signal handler on mule N
  - `mule`/`mule0` - run the signal handler on the first available mule
  - `spooler` - run the signal on the first available spooler
  - `farmN`/`farm_XXX` - run the signal handler in the mule farm N or named XXX

Raises **ValueError** – If unable to add timer.

uwsgiconf.runtime.scheduling.**register_timer_ms**(*period: int*, ***, *target: Union[str, int, uwsgiconf.runtime.signals.Signal] = None*) → Union[Callable, bool]

Add a millisecond resolution timer.

```
@register_timer_ms(300)
def repeat():
    do()
```

Parameters

- **period** – The interval (milliseconds) at which the signal is raised.
- **target** – Existing signal to raise or Signal Target to register signal implicitly.

  Available targets:

  - `workers` - run the signal handler on all the workers
  - `workerN` - run the signal handler only on worker N
  - `worker`/`worker0` - run the signal handler on the first available worker
  - `active-workers` - run the signal handlers on all the active [non-cheaped] workers
  - `mules` - run the signal handler on all of the mules
  - `muleN` - run the signal handler on mule N
  - `mule`/`mule0` - run the signal handler on the first available mule
  - `spooler` - run the signal on the first available spooler
  - `farmN`/`farm_XXX` - run the signal handler in the mule farm N or named XXX

Raises **ValueError** – If unable to add timer.

uwsgiconf.runtime.scheduling.**register_cron**(***, *weekday: Union[str, int] = None*, *month: Union[str, int] = None*, *day: Union[str, int] = None*, *hour: Union[str, int] = None*, *minute: Union[str, int] = None*, *target: Union[str, int, uwsgiconf.runtime.signals.Signal] = None*) → Union[Callable, bool]

Adds cron. The interface to the uWSGI signal cron facility.

```
@register_cron(hour=-3)   # Every 3 hours.
def repeat():
    do()
```

Note: Arguments work similarly to a standard crontab, but instead of "*", use -1, and instead of "/2", "/3", etc. use -2 and -3, etc.

Note: Periods - rules like hour='10-18/2' (from 10 till 18 every 2 hours) - are allowed, but they are emulated by uwsgiconf. Use strings to define periods.

Keep in mind, that your actual function will be wrapped into another one, which will check whether it is time to call your function.

Parameters

- **weekday** – Day of a the week number. Defaults to *each*. 0 - Sunday 1 - Monday 2 - Tuesday 3 - Wednesday 4 - Thursday 5 - Friday 6 - Saturday
- **month** – Month number 1-12. Defaults to *each*.
- **day** – Day of the month number 1-31. Defaults to *each*.
- **hour** – Hour 0-23. Defaults to *each*.
- **minute** – Minute 0-59. Defaults to *each*.
- **target** – Existing signal to raise or Signal Target to register signal implicitly.

  Available targets:

  - `workers` - run the signal handler on all the workers
  - `workerN` - run the signal handler only on worker N
  - `worker`/`worker0` - run the signal handler on the first available worker
  - `active-workers` - run the signal handlers on all the active [non-cheaped] workers
  - `mules` - run the signal handler on all of the mules
  - `muleN` - run the signal handler on mule N
  - `mule`/`mule0` - run the signal handler on the first available mule
  - `spooler` - run the signal on the first available spooler
  - `farmN`/`farm_XXX` - run the signal handler in the mule farm N or named XXX

Raises **ValueError** – If unable to add cron rule.

3.7.12 Signals

class uwsgiconf.runtime.signals.**SignalDescription**(*num*, *target*, *func*)
Registered signal information.

func
Alias for field number 2

num
Alias for field number 0

target
Alias for field number 1

uwsgiconf.runtime.signals.**registry_signals = []**
Registered signals.

uwsgiconf.runtime.signals.**get_available_num**() → int
Returns first available signal number.
Raises **UwsgiconfException** – If no signal is available.

uwsgiconf.runtime.signals.**get_last_received**() → uwsgiconf.runtime.signals.Signal
Get the last signal received.

class uwsgiconf.runtime.signals.**Signal**(*num: int = None*)
Represents uWSGI signal.

Warning: If you define a new function in worker1 and register it as a signal handler, only worker1 can run it. The best way to register signals is defining them in the master (.runtime.uwsgi.postfork_hooks.add), so all workers see them.

```
signal = Signal()

@signal.register_handler()
def somefunc():
    pass

# or the same:

@signal
def somefunc():
    pass
```

Parameters **num** (*int*) – Signal number (0-255).

Note: If not set it will be chosen automatically.

registered
Whether the signal is registered.

register_handler(*, *target: str = None*) → Callable
Decorator for a function to be used as a signal handler.

```
signal = Signal()

@signal.register_handler()
def somefunc():
    pass
```

Parameters **target** – Where this signal will be delivered to. Default: `worker`.

- `workers` - run the signal handler on all the workers
- `workerN` - run the signal handler only on worker N
- `worker`/`worker0` - run the signal handler on the first available worker
- `active-workers` - run the signal handlers on all the active [non-cheaped] workers
- `mules` - run the signal handler on all of the mules
- `muleN` - run the signal handler on mule N
- `mule`/`mule0` - run the signal handler on the first available mule
- `spooler` - run the signal on the first available spooler
- `farmN`/`farm_XXX` - run the signal handler in the mule farm N or named XXX
- http://uwsgi.readthedocs.io/en/latest/Signals.html#signals-targets

send(*, *remote: str = None*)
Sends the signal to master or remote.

When you send a signal, it is copied into the master's queue. The master will then check the signal table and dispatch the messages.

Parameters **remote** – Remote address.

Raises

- **ValueError** – If remote rejected the signal.
- **OSError** – If unable to deliver to remote.

wait()

Waits for the given of any signal.

Block the process/thread/async core until a signal is received. Use signal_received to get the number of the signal received. If a registered handler handles a signal, signal_wait will be interrupted and the actual handler will handle the signal.

- http://uwsgi-docs.readthedocs.io/en/latest/Signals.html#signal-wait-and-signal-received

Raises **SystemError** – If something went wrong.

3.7.13 Spooler

```
my_spooler = Spooler.get_by_basename('myspooler')

# @Spooler.task() to  run on first available or to run on `my_spooler`:
@my_spooler.task(postpone=timedelta(seconds=1))
def run_me(a, b='c'):
    # We do:
    # * return True if task processed
    # * return None if task was ignored
    # * raise an exception to force task retry
    return True

# Now call this function as usual and it'll run in a spooler.
...
run_me('some', b='other')
...
```

uwsgiconf.runtime.spooler.**spooler_task_types = {'fcall': <class 'uwsgiconf.runtime.spooler**

Known task types handlers will store here runtime.

SpoolerTask heirs are automatically registered in runtime by SpoolerTask.__init_subclass__.

class uwsgiconf.runtime.spooler.**Spooler**(*name: str*)

Gives an access to uWSGI Spooler related functions.

> **Warning:** To use this helper one needs to configure spooler(s) in uWSGI config beforehand.

```
my_spooler = Spooler.get_by_basename('myspooler')

# @Spooler.task() to  run on first available or to run on `my_spooler`:
@my_spooler.task(postpone=timedelta(seconds=1))
def run_me(a, b='c'):
    ...
```

(continues on next page)

(continued from previous page)

```
# Now call this function as usual and it'll run in a spooler.
...
run_me('some', b='other')
...
```

task(*postpone=None*)

Decorator. Used to register a function which should be run in Spooler.

```
my_spooler = Spooler.get_by_basename('myspooler')

# @Spooler.task() to  run on first available or to run on `my_spooler`:
@my_spooler.task(postpone=timedelta(seconds=1))
def run_me(a, b='c'):
    ...
```

classmethod send_message_raw(*message: str*, ***, *spooler: Union[str, Spooler] = None*, *priority: int = None*, *postpone: Union[datetime.datetime, datetime.timedelta] = None*, *payload: Any = None*) → str

Sends a message to a spooler.

Parameters

- **message** – Message to pass using spooler.
- **spooler** – The spooler (id or directory) to use. Specify the ABSOLUTE path of the spooler that has to manage this task
- **priority** – Number. The priority of the message. Larger - less important.

  Warning: This works only if you enable *order_tasks* option in *spooler.set_basic_params()*.

- **postpone** – Postpone message processing till.
- **payload** – Object to pickle and pass within message.

classmethod get_spoolers() → List[uwsgiconf.runtime.spooler.Spooler]

Returns a list of registered spoolers.

classmethod get_by_basename(*name: str*) → Optional[uwsgiconf.runtime.spooler.Spooler]

Returns spooler object for a given directory name.

If there is more than one spooler with the same directory base name, the first one is returned.

If not found *None* is returned.

Parameters **name** – Directory base name. E.g.: 'mydir' to get spooler for '/somewhere/here/is/mydir'

classmethod get_pids() → List[int]

Returns a list of all spooler processes IDs.

classmethod set_period(*seconds: int*) → bool

Sets how often the spooler runs.

Parameters **seconds** –

classmethod get_tasks() → List[str]

Returns a list of spooler jobs (filenames in spooler directory).

classmethod read_task_file(*path: str*) → dict
Returns a spooler task information.

Parameters **path** – The relative or absolute path to the task to read.

class uwsgiconf.runtime.spooler.**TaskResult**(*result: Any = None*, *, *exception: Exception = None*)
Represents a task processing result.

class uwsgiconf.runtime.spooler.**ResultProcessed**(*result: Any = None*, *, *exception: Exception = None*)
Treat task as processed.

class uwsgiconf.runtime.spooler.**ResultSkipped**(*result: Any = None*, *, *exception: Exception = None*)
Treat task as skipped (ignored).

class uwsgiconf.runtime.spooler.**ResultRescheduled**(*result: Any = None*, *, *exception: Exception = None*)
Treat task as rescheduled (being due to retry).

class uwsgiconf.runtime.spooler.**SpoolerTask**(*name: str*, *message: str*, *payload: Any*)
Consolidates information for a spooler task.

mark_processed
alias of *ResultProcessed*

mark_skipped
alias of *ResultSkipped*

mark_rescheduled
alias of *ResultRescheduled*

process() → Union[uwsgiconf.runtime.spooler.TaskResult, bool, None]
Processes the task.

Supported results:

- *None* - mark as ignored (skipped)
- *TaskResult* - result type logic
- exception - mark to retry
- other - mark as processed

class uwsgiconf.runtime.spooler.**SpoolerFunctionCallTask**(*name: str*, *message: str*, *payload: Any*)
Function call type. Allows delegating function calls to spoolers.

process() → Union[uwsgiconf.runtime.spooler.TaskResult, bool, None]
Processes the task.

Supported results:

- *None* - mark as ignored (skipped)
- *TaskResult* - result type logic
- exception - mark to retry
- other - mark as processed

3.8 Configuration [Static]

3.8.1 Configuration and Section

Configuration and Section are two types you'll have to mainly deal with.

class uwsgiconf.config.**Section**(*name: str = None*, ***, *runtime_dir: str = None*, *project_name: str = None*, *strict_config: bool = None*, *style_prints: bool = False*, *embedded_plugins: Union[Callable, List[str]] = None*, ***kwargs*)

Configuration section.

Options within configuration section are gathered into groups:

- alarms
- caching
- master_process
- workers
- etc.

Next to all public methods of groups are for setting configuration parameters. Such methods return section object to allow chaining.

You can pass options group basic parameters into (the following are all the same):

- `set_basic_params()` as in `section.workers.set_basic_params(count=3)`
- `__call__` as in `section.workers(count=3)`
- section initializer using *params_* prefixed group name:

```
Section(
    params_workers=dict(count=3),
)
```

Parameters

- **name** – Configuration section name.
- **runtime_dir** – Directory to store runtime files. See `.replace_placeholders()`

  Note: This can be used to store PID files, sockets, master FIFO, etc.

- **project_name** – Project name (alias) to be used to differentiate projects. See `.replace_placeholders()`.
- **strict_config** – Enable strict configuration parsing. If any unknown option is encountered in a configuration file, an error is shown and uWSGI quits.

  To use placeholder variables when using strict mode, use the `set-placeholder` option.

- **style_prints** – Enables styling (e.g. colouring) for `print_` family methods. Could be nice for console and distracting in logs.
- **embedded_plugins** – List of embedded plugins. Plugins from that list will be considered already loaded so uwsgiconf won't instruct uWSGI to load it if required.

  See `.embedded_plugins_presets` for shortcuts.

  Note:

  - If you installed uWSGI using PyPI package there should already be basic plugins embedded.
  - If using Ubuntu distribution you have to install plugins as separate packages.

  - http://uwsgi-docs.readthedocs.io/en/latest/BuildSystem.html#plugins-and-uwsgiplugin-py

alarms
alias of *uwsgiconf.options.alarms.Alarms*

applications
alias of *uwsgiconf.options.applications.Applications*

caching
alias of *uwsgiconf.options.caching.Caching*

cheapening
alias of *uwsgiconf.options.workers_cheapening.Cheapening*

empire
alias of *uwsgiconf.options.empire.Empire*

locks
alias of *uwsgiconf.options.locks.Locks*

logging
alias of *uwsgiconf.options.logging.Logging*

main_process
alias of *uwsgiconf.options.main_process.MainProcess*

master_process
alias of *uwsgiconf.options.master_process.MasterProcess*

monitoring
alias of *uwsgiconf.options.monitoring.Monitoring*

networking
alias of *uwsgiconf.options.networking.Networking*

queue
alias of *uwsgiconf.options.queue.Queue*

routing
alias of *uwsgiconf.options.routing.Routing*

spooler
alias of *uwsgiconf.options.spooler.Spooler*

statics
alias of *uwsgiconf.options.statics.Statics*

subscriptions
alias of *uwsgiconf.options.subscriptions.Subscriptions*

workers
alias of *uwsgiconf.options.workers.Workers*

python
alias of *uwsgiconf.options.python.Python*

class embedded_plugins_presets
These are plugin presets that can be used as `embedded_plugins` values.

BASIC = ['ping', 'cache', 'nagios', 'rrdtool', 'carbon', 'rpc', 'corerouter', 'fas
Basic set of embedded plugins. This set is used in uWSGI package from PyPI.

static PROBE(*uwsgi_binary: str = None*)
This preset allows probing real uWSGI to get actual embedded plugin list.

replace_placeholders(*value: Union[str, List[str], None]*) → Union[str, List[str], None]
Replaces placeholders that can be used e.g. in filepaths.

Supported placeholders:

- {project_runtime_dir}
- {project_name}
- {runtime_dir}

Parameters **value** –

project_name
Project name (alias) to be used to differentiate projects. See `.replace_placeholders()`.

get_runtime_dir(*, *default: bool = True*) → str
Directory to store runtime files. See `.replace_placeholders()`

Note: This can be used to store PID files, sockets, master FIFO, etc.

Parameters **default** – Whether to return [system] default if not set.

set_runtime_dir(*value*) → TypeSection
Sets user-defined runtime directory value.

Parameters **value** (*str*) –

as_configuration(***kwargs*) → uwsgiconf.config.Configuration
Returns configuration object including only one (this very) section.

Parameters **kwargs** – Configuration objects initializer arguments.

print_plugins() → TypeSection
Print out enabled plugins.

print_stamp() → TypeSection
Prints out a stamp containing useful information, such as what and when has generated this configuration.

print_out(*value: Any*, *, *indent: str = None*, *format_options: Union[dict*, *str] = None*, *asap: bool = False*) → TypeSection
Prints out the given value.

Parameters

- **value** –
- **indent** –
- **format_options** – text color
- **asap** – Print as soon as possible.

print_variables() → TypeSection
Prints out magic variables available in config files alongside with their values and descriptions. May be useful for debugging.

http://uwsgi-docs.readthedocs.io/en/latest/Configuration.html#magic-variables

set_plugins_params(**, plugins: Union[List[str], List[uwsgiconf.base.OptionsGroup], str, uwsgiconf.base.OptionsGroup] = None, search_dirs: Union[str, List[str]] = None, autoload: bool = None, required: bool = False*) → TypeSection
Sets plugin-related parameters.

Parameters

- **plugins** – uWSGI plugins to load
- **search_dirs** – Directories to search for uWSGI plugins.
- **autoload** – Try to automatically load plugins when unknown options are found.
- **required** – Load uWSGI plugins and exit on error.

set_fallback(*target: Union[str, Section]*) → TypeSection
Sets a fallback configuration for section.

Re-exec uWSGI with the specified config when exit code is 1.

Parameters **target** – File path or Section to include.

set_placeholder(*key: str*, *value: str*) → TypeSection
Placeholders are custom magic variables defined during configuration time.

Note: These are accessible, like any uWSGI option, in your application code via `.runtime.platform.uwsgi.config`.

Parameters

- **key** –
- **value** –

env(*key: str*, *value: Any = None*, ***, *unset: bool = False*, *asap: bool = False*, *update_local: bool = False*) → TypeSection
Processes (sets/unsets) environment variable.

If is not given in *set* mode value will be taken from current env.

Parameters

- **key** –
- **value** –
- **unset** – Whether to unset this variable.
- **asap** – If True env variable will be set as soon as possible.
- **update_local** – Whether we need to set this value for local environment too. This could be useful in embedded mode.

include(*target: Union[Section, List[Section], str, List[str]]*) → TypeSection
Includes target contents into config.

Parameters **target** – File path or Section to include.

classmethod derive_from(*section: TypeSection*, *, *name: str = None*) → TypeSection
Creates a new section based on the given.

Parameters

- **section** – Section to derive from,
- **name** – New section name.

class vars
The following variables also known as magic variables could be used as option values where appropriate.

- http://uwsgi-docs.readthedocs.io/en/latest/Configuration.html#magic-variables

VERSION = '%V'
uWSGI version number

FORMAT_ESCAPE = '%['
ANSI escape 033. useful for printing colors

CONF_CURRENT_SECTION = '%x'
The current section identifier, eg. conf.ini:section.

CONF_NAME_ORIGINAL = '%o'
The original conf filename, as specified on the command line

TIMESTAMP_STARTUP_S = '%t'
Unix time s, gathered at instance startup.

TIMESTAMP_STARTUP_MS = '%T'
Unix time ms, gathered at instance startup

DIR_VASSALS = '%v'
Vassals directory - pwd.

HOST_NAME = '%h'
Host name.

CPU_CORES = '%k'
Detected CPU count.

USER_ID = '%u'
User ID.

USER_NAME = '%U'
User name.

GROUP_ID = '%g'
Use group ID.

GROUP_NAME = '%G'
Use group name.

classmethod get_descriptions() → Dict[str, str]
Returns variable to description mapping.

classmethod bootstrap(*dsn: Union[str, List[str]]*, *, *allow_shared_sockets: bool = None*, ***init_kwargs*) → TypeSection
Constructs a section object performing it's basic (default) configuration.

Parameters

- **dsn** – Data source name, e.g: * http://127.0.0.1:8000 * https://127.0.0.1:443?cert=/here/there.crt&key=/that/my.key

Note: Some schemas: fastcgi, http, https, raw, scgi, shared, udp, uwsgi, suwsgi, zeromq

- **allow_shared_sockets** – Allows using shared sockets to bind to privileged ports. If not provided automatic mode is enabled: shared are allowed if current user is not root.
- **init_kwargs** – Additional initialization keyword arguments accepted by section type.

class uwsgiconf.config.**Configuration**(*sections: List[uwsgiconf.config.Section] = None*, ***, *autoinclude_sections: bool = False*, *alias: str = None*)

Configuration is comprised from one or more Sections and could be represented in format natively supported by uWSGI.

Parameters

- **sections** – If not provided, empty section will be automatically generated.
- **autoinclude_sections** – Whether to include in the first sections all subsequent sections.
- **alias** – Configuration alias. This will be used in `tofile` as file name.

format(***, *do_print: bool = False*, *stamp: bool = True*, *formatter: str = 'ini'*) → Union[str, List[str]]

Applies formatting to configuration.

Parameters

- **do_print** – Whether to print out formatted config.
- **stamp** – Whether to add stamp data to the first configuration section.
- **formatter** – Formatter alias to format options. Default: ini.

print_ini() → Union[str, List[str]]

Print out this configuration as .ini.

tofile(*filepath: Union[str, pathlib.Path] = None*) → str

Saves configuration into a file and returns its path.

Convenience method.

Parameters **filepath** – Filepath to save configuration into. If not provided a temporary file will be automatically generated.

uwsgiconf.config.**configure_uwsgi**(*configurator_func: Callable*) → Optional[List[uwsgiconf.config.Configuration]]

Allows configuring uWSGI using Configuration objects returned by the given configuration function.

Returns a list with detected configurations or `None` if called from within uWSGI (e.g. when trying to load WSGI application).

```
# In configuration module, e.g `uwsgicfg.py`

from uwsgiconf.config import configure_uwsgi

configure_uwsgi(get_configurations)
```

Parameters **configurator_func** – Function which return a list on configurations.

Raises **ConfigurationError** –

3.8.2 Alarms

Alarm Types

class uwsgiconf.options.alarm_types.**AlarmType**(*alias: str*, **args*, ***kwargs*)

class uwsgiconf.options.alarm_types.**AlarmCommand**(*alias: str*, ***, *command: str*)
Run a shell command, passing info into its stdin.

class uwsgiconf.options.alarm_types.**AlarmSignal**(*alias: str*, ***, *sig: int*)
Raise an uWSGI signal.

class uwsgiconf.options.alarm_types.**AlarmLog**(*alias: str*)
Print line into log.

class uwsgiconf.options.alarm_types.**AlarmMule**(*alias: str*, ***, *mule: int*)
Send info to a mule waiting for messages.

class uwsgiconf.options.alarm_types.**AlarmCurl**(*alias: str*, *url: str*, ***, *method: str = None*, *ssl: bool = None*, *ssl_insecure: bool = None*, *auth_user: str = None*, *auth_pass: str = None*, *timeout: int = None*, *conn_timeout: int = None*, *mail_from: str = None*, *mail_to: str = None*, *subject: str = None*)
Send info to a cURL-able URL.

class uwsgiconf.options.alarm_types.**AlarmXmpp**(*alias: str, *, jid: str, password: str, recipients: Union[str, List[str]]*)
Send info via XMPP/jabber.

class uwsgiconf.options.alarms.**Alarms**(**args*, ***kwargs*)
Alarms.

This subsystem allows the developer/sysadmin to "announce" special conditions of an app via various channels.
- http://uwsgi-docs.readthedocs.io/en/latest/AlarmSubsystem.html

class alarm_types
Alarm types available for `.register_alarm()`.

command
alias of *uwsgiconf.options.alarm_types.AlarmCommand*

curl
alias of *uwsgiconf.options.alarm_types.AlarmCurl*

log
alias of *uwsgiconf.options.alarm_types.AlarmLog*

mule
alias of *uwsgiconf.options.alarm_types.AlarmMule*

signal
alias of *uwsgiconf.options.alarm_types.AlarmSignal*

xmpp
alias of *uwsgiconf.options.alarm_types.AlarmXmpp*

set_basic_params(***, *msg_size: int = None*, *cheap: bool = None*, *anti_loop_timeout: int = None*)

Parameters

- **msg_size** – Set the max size of an alarm message in bytes. Default: 8192.

- **cheap** – Use main alarm thread rather than create dedicated threads for curl-based alarms
- **anti_loop_timeout** – Tune the anti-loop alarm system. Default: 3 seconds.

print_alarms()
Print out enabled alarms.

register_alarm(*alarm: List[uwsgiconf.options.alarm_types.AlarmType]*)
Register (create) an alarm.

Parameters **alarm** – Alarm.

alarm_on_log(*alarm: List[uwsgiconf.options.alarm_types.AlarmType], matcher: str, *, skip: bool = False*)
Raise (or skip) the specified alarm when a log line matches the specified regexp.

Parameters

- **alarm** – Alarm.
- **matcher** – Regular expression to match log line.
- **skip** –

alarm_on_fd_ready(*alarm: List[uwsgiconf.options.alarm_types.AlarmType], *, fd: str, message: str, byte_count: int = None*)
Triggers the alarm when the specified file descriptor is ready for read.

This is really useful for integration with the Linux eventfd() facility. Pretty low-level and the basis of most of the alarm plugins.

- http://uwsgi-docs.readthedocs.io/en/latest/Changelog-1.9.7.html#alarm-fd

Parameters

- **alarm** – Alarm.
- **fd** – File descriptor.
- **message** – Message to send.
- **byte_count** – Files to read. Default: 1 byte.

Note: For `eventfd` set 8.

alarm_on_queue_full(*alarm: List[uwsgiconf.options.alarm_types.AlarmType]*)
Raise the specified alarm when the socket backlog queue is full.

Parameters **alarm** – Alarm.

alarm_on_segfault(*alarm: List[uwsgiconf.options.alarm_types.AlarmType]*)
Raise the specified alarm when the segmentation fault handler is executed.

Sends a backtrace.

Parameters **alarm** – Alarm.

3.8.3 Applications

class uwsgiconf.options.applications.**Applications**(**args*, ***kwargs*)
Applications.

set_basic_params(*, *exit_if_none: bool = None*, *max_per_worker: int = None*, *single_interpreter: bool = None*, *no_default: bool = None*, *manage_script_name: bool = None*)

Parameters

- **exit_if_none** – Exit if no app can be loaded.
- **max_per_worker** – Set the maximum number of per-worker applications.
- **single_interpreter** – Do not use multiple interpreters (where available). Some of the supported languages (such as Python) have the concept of "multiple interpreters". By default every app is loaded in a new python interpreter (that means a pretty-well isolated namespace for each app). If you want all of the app to be loaded in the same python vm, use the this option.
- **no_default** – Do not automatically fallback to default app. By default, the first loaded app is mounted as the "default one". That app will be served when no mountpoint matches.
- **manage_script_name** – You can to instruct uWSGI to map specific apps in the so called "mountpoint" and rewrite SCRIPT_NAME and PATH_INFO automatically. See .mount(). The WSGI standard dictates that SCRIPT_NAME is the variable used to select a specific application.

mount(*mountpoint: str*, *app: str*, *, *into_worker: bool = False*)

Load application under mountpoint.

Example:

- .mount('', 'app0.py') – Root URL part
- .mount('/app1', 'app1.py') – URL part
- .mount('/pinax/here', '/var/www/pinax/deploy/pinax.wsgi')
- .mount('the_app3', 'app3.py') – Variable value: application alias (can be set by `UWSGI_APPID`)
- .mount('example.com', 'app2.py') – Variable value: Hostname (variable set in nginx)

- http://uwsgi-docs.readthedocs.io/en/latest/Nginx.html#hosting-multiple-apps-in-the-same-process-aka-managing-sc

Parameters

- **mountpoint** – URL part, or variable value.

Note: In case of URL part you may also want to set `manage_script_name` basic param to `True`.

Warning: In case of URL part a trailing slash may case problems in some cases (e.g. with Django based projects).

- **app** – App module/file.
- **into_worker** – Load application under mountpoint in the specified worker or after workers spawn.

switch_into_lazy_mode(*, *affect_master: bool = None*)
Load apps in workers instead of master.

This option may have memory usage implications as Copy-on-Write semantics can not be used.

Note: Consider using `touch_chain_reload` option in `workers` basic params for lazy apps reloading.

Parameters **affect_master** – If **True** only workers will be reloaded by uWSGI's reload signals; the master will remain alive.

> **Warning:** uWSGI configuration changes are not picked up on reload by the master.

3.8.4 Caching

class uwsgiconf.options.caching.**Caching**(**args*, ***kwargs*)
Caching.

uWSGI includes a very fast, all-in-memory, zero-IPC, SMP-safe, constantly-optimizing, highly-tunable, key-value store simply called "the caching framework".

A single uWSGI instance can create an unlimited number of "caches" each one with different setup and purpose.
- http://uwsgi-docs.readthedocs.io/en/latest/Caching.html
- http://uwsgi-docs.readthedocs.io/en/latest/tutorials/CachingCookbook.html

set_basic_params(*, *no_expire: bool = None*, *expire_scan_interval: int = None*, *report_freed: bool = None*)

Parameters

- **no_expire** – Disable auto sweep of expired items. Since uWSGI 1.2, cache item expiration is managed by a thread in the master process, to reduce the risk of deadlock. This thread can be disabled (making item expiry a no-op) with the this option.
- **expire_scan_interval** – Set the frequency (in seconds) of cache sweeper scans. Default: 3.
- **report_freed** – Constantly report the cache item freed by the sweeper.

> **Warning:** Use only for debug.

add_item(*key: str*, *value: str*, *, *cache_name: str = None*)
Add an item into the given cache.

This is a commodity option (mainly useful for testing) allowing you to store an item in a uWSGI cache during startup.

Parameters

- **key** –
- **value** –
- **cache_name** – If not set, default will be used.

add_file(*filepath: Union[str, pathlib.Path], *, gzip: bool = False, cache_name: str = None*)
Load a static file in the cache.

Note: Items are stored with the filepath as is (relative or absolute) as the key.

Parameters

- **filepath** –
- **gzip** – Use gzip compression.
- **cache_name** – If not set, default will be used.

add_cache(*name: str*, ***, *max_items: int*, *no_expire: bool = None*, *store: str = None*, *store_sync_interval: int = None*, *store_delete: bool = None*, *hash_algo: str = None*, *hash_size: int = None*, *key_size: int = None*, *udp_clients: Union[str, List[str]] = None*, *udp_servers: Union[str, List[str]] = None*, *block_size: int = None*, *block_count: int = None*, *sync_from: Union[str, List[str]] = None*, *mode_bitmap: bool = None*, *use_lastmod: bool = None*, *full_silent: bool = None*, *full_purge_lru: bool = None*)
Creates cache. Default mode: single block.

Note: This uses new generation `cache2` option available since uWSGI 1.9.

Note: When at least one cache is configured without `full_purge_lru` and the master is enabled a thread named "the cache sweeper" is started. Its main purpose is deleting expired keys from the cache. If you want auto-expiring you need to enable the master.

Parameters

- **name** – Set the name of the cache. Must be unique in an instance.
- **max_items** – Set the maximum number of cache items.

  Note: Effective number of items is **max_items - 1** - the first item of the cache is always internally used as "NULL/None/undef".

- **no_expire** – If `True` cache items won't expire even if instructed to do so by cache set method.
- **store** – Set the filename for the persistent storage. If it doesn't exist, the system assumes an empty cache and the file will be created.
- **store_sync_interval** – Set the number of seconds after which msync() is called to flush memory cache on disk when in persistent mode. By default it is disabled leaving the decision-making to the kernel.
- **store_delete** – uWSGI, by default, will not start if a cache file exists and the store file does not match the configured items/blocksize. Setting this option will make uWSGI delete the existing file upon mismatch and create a new one.
- **hash_algo** – Set the hash algorithm used in the hash table. Current options are:
  - djb33x (default)

  - murmur2

- **hash_size** – This is the size of the hash table in bytes. Generally 65536 (the default) is a good value.

  Note: Change it only if you know what you are doing or if you have a lot of collisions in your cache.

- **key_size** – Set the maximum size of a key, in bytes. Default: 2048.
- **udp_clients** – List of UDP servers which will receive UDP cache updates.
- **udp_servers** – List of UDP addresses on which to bind the cache to wait for UDP updates.
- **block_size** – Set the size (in bytes) of a single block.

  Note: It's a good idea to use a multiple of 4096 (common memory page size).

- **block_count** – Set the number of blocks in the cache. Useful only in bitmap mode, otherwise the number of blocks is equal to the maximum number of items.
- **sync_from** – List of uWSGI addresses which the cache subsystem will connect to for getting a full dump of the cache. It can be used for initial cache synchronization. The first node sending a valid dump will stop the procedure.
- **mode_bitmap** – Enable (more versatile but relatively slower) bitmap mode.

  http://uwsgi-docs.readthedocs.io/en/latest/Caching.html#single-block-faster-vs-bitmaps-slower

  Warning: Considered production ready only from uWSGI 2.0.2.

- **use_lastmod** – Enabling will update last_modified_at timestamp of each cache on every cache item modification. Enable it if you want to track this value or if other features depend on it. This value will then be accessible via the stats socket.
- **full_silent** – By default uWSGI will print warning message on every cache set operation if the cache is full. To disable this warning set this option.

  Note: Available since 2.0.4.

- **full_purge_lru** – Allows the caching framework to evict Least Recently Used (LRU) item when you try to add new item to cache storage that is full.

  Note: `no_expire` argument will be ignored.

3.8.5 Empire

class uwsgiconf.options.empire.**Empire**(**args*, ***kwargs*)

Emperor and his vassals.

If you need to deploy a big number of apps on a single server, or a group of servers, the Emperor mode is just the ticket.

- http://uwsgi-docs.readthedocs.io/en/latest/Emperor.html

set_emperor_params(*, *vassals_home=None*, *name=None*, *scan_interval=None*, *pid_file=None*, *spawn_asap=None*, *stats_address=None*, *trigger_socket=None*, *links_no_follow=None*)

Note: The emperor should generally not be run with master, unless master features like advanced logging are specifically needed.

Note: The emperor should generally be started at server boot time and left alone, not reloaded/restarted except for uWSGI upgrades; emperor reloads are a bit drastic, reloading all vassals at once. Instead vassals should be reloaded individually when needed, in the manner of the imperial monitor in use.

Parameters

- **vassals_home** (*str|list[str]*) – Set vassals home and enable Emperor mode.
- **name** (*str*) – Set the Emperor process name.
- **scan_interval** (*int*) – Set the Emperor scan frequency. Default: 3 seconds.
- **pid_file** (*str*) – Write the Emperor pid in the specified file.
- **spawn_asap** (*bool*) – Spawn the Emperor as soon as possible.
- **stats_address** (*str*) – Run the Emperor stats server on specified address.
- **trigger_socket** (*str*) – Enable the Emperor trigger socket.
- **links_no_follow** (*bool*) – Do not follow symlinks when checking for mtime.

print_monitors()

Print out enabled imperial monitors.

set_emperor_command_params(*command_socket=None*, *, *wait_for_command=None*, *wait_for_command_exclude=None*)

Emperor commands related parameters.

- http://uwsgi-docs.readthedocs.io/en/latest/tutorials/EmperorSubscriptions.html

Parameters

- **command_socket** (*str*) – Enable the Emperor command socket. It is a channel allowing external process to govern vassals.
- **wait_for_command** (*bool*) – Always wait for a 'spawn' Emperor command before starting a vassal.
- **wait_for_command_exclude** (*str|list[str]*) – Vassals that will ignore `wait_for_command`.

set_vassals_wrapper_params(*, *wrapper=None*, *overrides=None*, *fallbacks=None*)

Binary wrapper for vassals parameters.

Parameters

- **wrapper** (*str*) – Set a binary wrapper for vassals.
- **overrides** (*str|list[str]*) – Set a binary wrapper for vassals to try before the default one
- **fallbacks** (*str|list[str]*) – Set a binary wrapper for vassals to try as a last resort. Allows you to specify an alternative binary to execute when running a vassal and the default binary_path is not found (or returns an error).

set_throttle_params(*, *level=None*, *level_max=None*)

Throttling options.

- http://uwsgi-docs.readthedocs.io/en/latest/Emperor.html#throttling
- http://uwsgi-docs.readthedocs.io/en/latest/Emperor.html#loyalty

Parameters

- **level** (*int*) – Set throttling level (in milliseconds) for bad behaving vassals. Default: 1000.
- **level_max** (*int*) – Set maximum throttling level (in milliseconds) for bad behaving vassals. Default: 3 minutes.

set_tolerance_params(*, *for_heartbeat=None*, *for_cursed_vassals=None*)

Various tolerance options.

Parameters

- **for_heartbeat** (*int*) – Set the Emperor tolerance about heartbeats.
  - http://uwsgi-docs.readthedocs.io/en/latest/Emperor.html#heartbeat-system
- **for_cursed_vassals** (*int*) – Set the Emperor tolerance about cursed vassals.
  - http://uwsgi-docs.readthedocs.io/en/latest/Emperor.html#blacklist-system

set_mode_tyrant_params(*enable=None*, *, *links_no_follow=None*, *use_initgroups=None*)

Tyrant mode (secure multi-user hosting).

In Tyrant mode the Emperor will run the vassal using the UID/GID of the vassal configuration file.

- http://uwsgi-docs.readthedocs.io/en/latest/Emperor.html#tyrant-mode-secure-multi-user-hosting

Parameters

- **enable** – Puts the Emperor in Tyrant mode.
- **links_no_follow** (*bool*) – Do not follow symlinks when checking for uid/gid in Tyrant mode.
- **use_initgroups** (*bool*) – Add additional groups set via initgroups() in Tyrant mode.

set_mode_broodlord_params(*zerg_count=None*, *, *vassal_overload_sos_interval=None*, *vassal_queue_items_sos=None*)

This mode is a way for a vassal to ask for reinforcements to the Emperor.

Reinforcements are new vassals spawned on demand generally bound on the same socket.

> **Warning:** If you are looking for a way to dynamically adapt the number of workers of an instance, check the Cheaper subsystem - adaptive process spawning mode.
>
> *Broodlord mode is for spawning totally new instances.*

Parameters

- **zerg_count** (*int*) – Maximum number of zergs to spawn.
- **vassal_overload_sos_interval** (*int*) – Ask emperor for reinforcement when overloaded. Accepts the number of seconds to wait between asking for a new reinforcements.
- **vassal_queue_items_sos** (*int*) – Ask emperor for sos if listen queue (backlog) has more items than the value specified

3.8.6 Locks

class uwsgiconf.options.locks.**Locks**(**args*, ***kwargs*)

Locks.

- http://uwsgi.readthedocs.io/en/latest/Locks.html

set_basic_params(*, *count: int = None*, *thunder_lock: bool = None*, *lock_engine: str = None*)

Parameters

- **count** – Create the specified number of shared locks.
- **thunder_lock** – Serialize accept() usage (if possible) Could improve performance on Linux with robust pthread mutexes.

  http://uwsgi.readthedocs.io/en/latest/articles/SerializingAccept.html
- **lock_engine** – Set the lock engine.

  Example:

  - ipcsem

set_ipcsem_params(*, *ftok: str = None*, *persistent: bool = None*)

Sets ipcsem lock engine params.

Parameters

- **ftok** – Set the ipcsem key via ftok() for avoiding duplicates.
- **persistent** – Do not remove ipcsem's on shutdown.

lock_file(*fpath: Union[str, pathlib.Path]*, **, after_setup: bool = False, wait: bool = False*)

Locks the specified file.

Parameters

- **fpath** – File path.
- **after_setup** – True - after logging/daemon setup False - before starting
- **wait** – True - wait if locked False - exit if locked

3.8.7 Logging

Loggers

class uwsgiconf.options.logging_loggers.**Logger**(*alias*, **args*)

class uwsgiconf.options.logging_loggers.**LoggerFile**(*filepath: Union[str, pathlib.Path], *, alias=None*)

Allows logging into files.

Parameters

- **filepath** (*str*) – File path.
- **alias** (*str*) – Logger alias.

class uwsgiconf.options.logging_loggers.**LoggerFileDescriptor**(*fd: int, *, alias=None*)

Allows logging using file descriptor.

Parameters

- **fd** (*str*) – File descriptor.
- **alias** (*str*) – Logger alias.

class uwsgiconf.options.logging_loggers.**LoggerStdIO**(**, alias=None*)

Allows logging stdio.

Parameters **alias** (*str*) – Logger alias.

class uwsgiconf.options.logging_loggers.**LoggerSocket**(*addr_or_path: Union[str, pathlib.Path], *, alias=None*)

Allows logging into UNIX and UDP sockets.

Parameters

- **addr_or_path** (*str*) – Remote address or filepath.

  Examples:

  - /tmp/uwsgi.logsock
  - 192.168.173.19:5050

- **alias** (*str*) – Logger alias.

class uwsgiconf.options.logging_loggers.**LoggerSyslog**(**, app_name=None, facility=None, alias=None*)

Allows logging into Unix standard syslog.

Parameters

- **app_name** (*str*) –
- **facility** (*str*) –
  - https://en.wikipedia.org/wiki/Syslog#Facility
- **alias** (*str*) – Logger alias.

class uwsgiconf.options.logging_loggers.**LoggerRsyslog**(**, app_name=None, host=None, facility=None, split=None, packet_size=None, alias=None*)

Allows logging into Unix standard syslog or a remote syslog.

Parameters

- **app_name** (*str*) –

- **host** (*str*) – Address (host and port) or UNIX socket path.
- **facility** (*str*) –
  - https://en.wikipedia.org/wiki/Syslog#Facility
- **split** (*bool*) – Split big messages into multiple chunks if they are bigger than allowed packet size. Default: `False`.
- **packet_size** (*int*) – Set maximum packet size for syslog messages. Default: 1024.

  Warning: using packets > 1024 breaks RFC 3164 (#4.1)

- **alias** (*str*) – Logger alias.

class uwsgiconf.options.logging_loggers.**LoggerRedis**(*, *host=None*, *command=None*, *prefix=None*, *alias=None*)

Allows logging into Redis.

Note: Consider using `dedicate_thread` param.

Parameters

- **host** (*str*) – Default: 127.0.0.1:6379
- **command** (*str*) – Command to be used. Default: publish uwsgi

  Examples:
  - publish foobar
  - rpush foo
- **prefix** (*str*) – Default: <empty>
- **alias** (*str*) – Logger alias.

class uwsgiconf.options.logging_loggers.**LoggerMongo**(*, *host=None*, *collection=None*, *node=None*, *alias=None*)

Allows logging into Mongo DB.

Note: Consider using `dedicate_thread` param.

Parameters

- **host** (*str*) – Default: 127.0.0.1:27017
- **collection** (*str*) – Command to be used. Default: uwsgi.logs
- **node** (*str*) – An identification string for the instance sending logs Default: <server hostname>
- **alias** (*str*) – Logger alias.

class uwsgiconf.options.logging_loggers.**LoggerZeroMq**(*connection_str*, *, *alias=None*)

Allows logging into ZeroMQ sockets.

Parameters

- **connection_str** (*str*) –

Examples:

- tcp://192.168.173.18:9191

- **alias** (*str*) – Logger alias.

Encoders

class uwsgiconf.options.logging_encoders.**Encoder**(**args*)

class uwsgiconf.options.logging_encoders.**EncoderPrefix**(*value*)
Add a raw prefix to each log msg.
Parameters **value** (*str*) – Value to be used as affix

class uwsgiconf.options.logging_encoders.**EncoderSuffix**(*value*)
Add a raw suffix to each log msg
Parameters **value** (*str*) – Value to be used as affix

class uwsgiconf.options.logging_encoders.**EncoderNewline**(**args*)
Add a newline char to each log msg.

class uwsgiconf.options.logging_encoders.**EncoderGzip**(**args*)
Compress each msg with gzip (requires zlib).

class uwsgiconf.options.logging_encoders.**EncoderCompress**(**args*)
Compress each msg with zlib compress (requires zlib).

class uwsgiconf.options.logging_encoders.**TimeFormatter**(*fmt: str*)
Allows user-defined time value formatting.
Parameters **fmt** – Time value format Format string (as for *strftime*)

Aliases:

- **iso - ISO 8601: %Y-%m-%dT%H:%M:%S%z** 2020-11-29T04:44:08+0000

class uwsgiconf.options.logging_encoders.**EncoderFormat**(*template*)
Apply the specified format to each log msg.
Parameters **template** (*str*) – Template string. Available variables are listed in FormatEncoder.Vars.

class vars
Variables available to use.

MESSAGE = '${msg}'
Raw log message (newline stripped).

MESSAGE_NEWLINE = '${msgnl}'
Raw log message (with newline).

TIME = '${unix}'
Current unix time.

TIME_US = '${micros}'
Current unix time in microseconds.

TIME_MS = '${millis}'
Current unix time in milliseconds.

TIME_FORMAT
Current time in user-defined format.

alias of *TimeFormatter*

class uwsgiconf.options.logging_encoders.**EncoderJson**(*template*)
Apply the specified format to each log msg with each variable json escaped.
Parameters **template** (*str*) – Template string. Available variables are listed in FormatEncoder.Vars.

class uwsgiconf.options.logging.**Var**(*name: str*)

class uwsgiconf.options.logging.**VarMetric**(*name: str*)

class uwsgiconf.options.logging.**VarRequestVar**(*name*)

class uwsgiconf.options.logging.**Logging**(**args*, ***kwargs*)
Logging.
- http://uwsgi.readthedocs.io/en/latest/Logging.html
- http://uwsgi-docs.readthedocs.io/en/latest/LogFormat.html

class loggers
Loggers available for add_logger().

file
alias of *uwsgiconf.options.logging_loggers.LoggerFile*

fd
alias of *uwsgiconf.options.logging_loggers.LoggerFileDescriptor*

stdio
alias of *uwsgiconf.options.logging_loggers.LoggerStdIO*

mongo
alias of *uwsgiconf.options.logging_loggers.LoggerMongo*

redis
alias of *uwsgiconf.options.logging_loggers.LoggerRedis*

socket
alias of *uwsgiconf.options.logging_loggers.LoggerSocket*

syslog
alias of *uwsgiconf.options.logging_loggers.LoggerSyslog*

rsyslog
alias of *uwsgiconf.options.logging_loggers.LoggerRsyslog*

zeromq
alias of *uwsgiconf.options.logging_loggers.LoggerZeroMq*

class encoders
Loggers available for add_logger_encoder().

compress
alias of *uwsgiconf.options.logging_encoders.EncoderCompress*

format
alias of *uwsgiconf.options.logging_encoders.EncoderFormat*

gzip
alias of *uwsgiconf.options.logging_encoders.EncoderGzip*

json
alias of *uwsgiconf.options.logging_encoders.EncoderJson*

newline
alias of *uwsgiconf.options.logging_encoders.EncoderNewline*

prefix
alias of *uwsgiconf.options.logging_encoders.EncoderPrefix*

suffix
alias of *uwsgiconf.options.logging_encoders.EncoderSuffix*

set_basic_params(*, *no_requests=None*, *template=None*, *memory_report=None*, *prefix=None*, *prefix_date=None*, *apply_strftime=None*, *response_ms=None*, *ip_x_forwarded=None*)

Parameters

- **no_requests** (*bool*) – Disable requests logging - only uWSGI internal messages and errors will be logged.
- **template** (*str*) – Set advanced format for request logging. This template string can use variables from `Logging.Vars`.
- **prefix** (*str*) – Prefix log items with a string.
- **prefix_date** (*str|bool*) – Prefix log items with date string.

  Note: This can be `True` or contain formatting placeholders (e.g. %Y-%m-%d %H:%M:%S) if used with `apply_strftime`.

- **memory_report** (*int*) – Enable memory report. * **1** - basic (default); * **2** - uss/pss (Linux only)
- **apply_strftime** (*bool*) – Apply strftime to dates in log entries. E.g. `prefix_date` can contain format placeholders. See also `vars.REQ_START_FORMATTED`.
- **response_ms** (*bool*) – Report response time in microseconds instead of milliseconds.
- **ip_x_forwarded** (*bool*) – Use the IP from X-Forwarded-For header instead of REMOTE_ADDR. Used when uWSGI is run behind multiple proxies.

log_into(*target*, *, *before_priv_drop=True*)
Simple file or UDP logging.

Note: This doesn't require any Logger plugin and can be used if no log routing is required.

Parameters

- **target** (*str*) – Filepath or UDP address.
- **before_priv_drop** (*bool*) – Whether to log data before or after privileges drop.

set_file_params(*, *reopen_on_reload=None*, *trucate_on_statup=None*, *max_size=None*, *rotation_fname=None*, *touch_reopen=None*, *touch_rotate=None*, *owner=None*, *mode=None*)
Set various parameters related to file logging.

Parameters

- **reopen_on_reload** (*bool*) – Reopen log after reload.
- **trucate_on_statup** (*bool*) – Truncate log on startup.

- **max_size** (*int*) – Set maximum logfile size in bytes after which log should be rotated.
- **rotation_fname** (*str*) – Set log file name after rotation.
- **touch_reopen** (*str|list*) – Trigger log reopen if the specified file is modified/touched.

  Note: This can be set to a file touched by `postrotate` script of `logrotate` to implement rotation.

- **touch_rotate** (*str|list*) – Trigger log rotation if the specified file is modified/touched.
- **owner** (*str*) – Set owner chown() for logs.
- **mode** (*str*) – Set mode chmod() for logs.

set_filters(*, *include=None*, *exclude=None*, *write_errors=None*, *write_errors_tolerance=None*, *sigpipe=None*)

Set various log data filters.

Parameters

- **include** (*str|list*) – Show only log lines matching the specified regexp.

  Note: Requires enabled PCRE support.

- **exclude** (*str|list*) – Do not show log lines matching the specified regexp.

  Note: Requires enabled PCRE support.

- **write_errors** (*bool*) – Log (annoying) write()/writev() errors. Default: `True`.

  Note: If both this and `sigpipe` set to `False`, it's the same as setting `write-errors-exception-only` uWSGI option.

- **write_errors_tolerance** (*int*) – Set the maximum number of allowed write errors before exception is raised. Default: no tolerance.

  Note: Available for Python, Perl, PHP.

- **sigpipe** (*bool*) – Log (annoying) SIGPIPE. Default: `True`.

  Note: If both this and `write_errors` set to `False`, it's the same as setting `write-errors-exception-only` uWSGI option.

set_requests_filters(*, *slower=None*, *bigger=None*, *status_4xx=None*, *status_5xx=None*, *no_body=None*, *sendfile=None*, *io_errors=None*)

Set various log data filters.

Parameters

- **slower** (*int*) – Log requests slower than the specified number of milliseconds.
- **bigger** (*int*) – Log requests bigger than the specified size in bytes.
- **status_4xx** – Log requests with a 4xx response.
- **status_5xx** – Log requests with a 5xx response.
- **no_body** (*bool*) – Log responses without body.
- **sendfile** (*bool*) – Log sendfile requests.
- **io_errors** (*bool*) – Log requests with io errors.

set_master_logging_params(*enable=None*, ***, *dedicate_thread=None*, *buffer=None*, *sock_stream=None*, *sock_stream_requests_only=None*)

Sets logging params for delegating logging to master process.

Parameters

- **enable** (*bool*) – Delegate logging to master process. Delegate the write of the logs to the master process (this will put all of the logging I/O to a single process). Useful for system with advanced I/O schedulers/elevators.
- **dedicate_thread** (*bool*) – Delegate log writing to a thread.

  As error situations could cause the master to block while writing a log line to a remote server, it may be a good idea to use this option and delegate writes to a secondary thread.
- **buffer** (*int*) – Set the buffer size for the master logger in bytes. Bigger log messages will be truncated.
- **sock_stream** (*bool|tuple*) – Create the master logpipe as SOCK_STREAM.
- **sock_stream_requests_only** (*bool|tuple*) – Create the master requests logpipe as SOCK_STREAM.

print_loggers()

Print out available (built) loggers.

add_logger(*logger*, ***, *requests_only=False*, *for_single_worker=False*)

Set/add a common logger or a request requests only.

Parameters

- **logger** (*str|list|Logger|list[Logger]*) –
- **requests_only** (*bool*) – Logger used only for requests information messages.
- **for_single_worker** (*bool*) – Logger to be used in single-worker setup.

add_logger_route(*logger*, *matcher*, ***, *requests_only=False*)

Log to the specified named logger if regexp applied on log item matches.

Parameters

- **logger** (*str|list|Logger|list[Logger]*) – Logger to associate route with.
- **matcher** (*str*) – Regular expression to apply to log item.
- **requests_only** (*bool*) – Matching should be used only for requests information messages.

add_logger_encoder(*encoder*, *, *logger=None*, *requests_only=False*, *for_single_worker=False*)
Add an item in the log encoder or request encoder chain.

- http://uwsgi-docs.readthedocs.io/en/latest/LogEncoders.html

Note: Encoders automatically enable master log handling (see `.set_master_logging_params()`).

Note: For best performance consider allocating a thread for log sending with `dedicate_thread`.

Parameters

- **encoder** (*str|list|Encoder*) – Encoder (or a list) to add into processing.
- **logger** (*str|Logger*) – Logger apply associate encoders to.
- **requests_only** (*bool*) – Encoder to be used only for requests information messages.
- **for_single_worker** (*bool*) – Encoder to be used in single-worker setup.

class vars
Variables available for custom log formatting.

REQ_URI = '%(uri)'
REQUEST_URI from `wsgi_request` of the current request.

REQ_METHOD = '%(method)'
REQUEST_METHOD from `wsgi_request` of the current request.

REQ_REMOTE_USER = '%(user)'
REMOTE_USER from `wsgi_request` of the current request.

REQ_REMOTE_ADDR = '%(addr)'
REMOTE_ADDR from `wsgi_request` of the current request.

REQ_HTTP_HOST = '%(host)'
HTTP_HOST from `wsgi_request` of the current request.

REQ_SERVER_PROTOCOL = '%(proto)'
SERVER_PROTOCOL from `wsgi_request` of the current request.

REQ_USER_AGENT = '%(uagent)'
HTTP_USER_AGENT from `wsgi_request` of the current request.

REQ_REFERER = '%(referer)'
HTTP_REFERER from `wsgi_request` of the current request.

REQ_START_TS = '%(time)'
Timestamp of the start of the request. E.g.: 1512623650

REQ_START_CTIME = '%(ctime)'
Ctime of the start of the request. E.g.: Thu Dec 7 08:05:35 2017

REQ_START_UNIX_US = '%(tmsecs)'
Timestamp of the start of the request in milliseconds since the epoch.

Note: since 1.9.21

REQ_START_UNIX_MS = '%(tmicros)'
Timestamp of the start of the request in microseconds since the epoch.

Note: since 1.9.21

REQ_START_HUMAN = '%(ltime)'
Human-formatted (Apache style) request time.

REQ_START_FORMATTED = '%(ftime)'
Request time formatted with `apply_strftime`.

Note: Use `apply_strftime` and placeholders.

REQ_SIZE_BODY = '%(cl)'
Request content body size.

REQ_COUNT_VARS_CGI = '%(vars)'
Number of CGI vars in the request.

REQ_COUNT_ERR_READ = '%(rerr)'
Number of read errors for the request.

Note: since 1.9.21

REQ_COUNT_ERR_WRITE = '%(werr)'
Number of write errors for the request.

Note: since 1.9.21

REQ_COUNT_ERR = '%(ioerr)'
Number of write and read errors for the request.

Note: since 1.9.21

RESP_STATUS = '%(status)'
HTTP response status code.

RESP_TIME_US = '%(micros)'
Response time in microseconds. E.g.: 1512623650704

RESP_TIME_MS = '%(msecs)'
Response time in milliseconds. E.g.: 1512623650704413

RESP_SIZE = '%(size)'
Response body size + response headers size.

RESP_SIZE_HEADERS = '%(hsize)'
Response headers size.

RESP_SIZE_BODY = '%(rsize)'
Response body size.

RESP_COUNT_HEADERS = '%(headers)'
Number of generated response headers.

TIME_UNIX = '%(epoch)'
The current time in Unix format.

WORKER_PID = '%(pid)'
pid of the worker handling the request.

WORKER_ID = '%(wid)'
id of the worker handling the request.

ASYNC_SWITCHES = '%(switches)'
Number of async switches.

CORE = '%(core)'
The core running the request.

MEM_VSZ = '%(vsz)'
Address space/virtual memory usage (in bytes).

MEM_RSS = '%(rss)'
RSS memory usage (in bytes).

MEM_VSZ_MB = '%(vszM)'
Address space/virtual memory usage (in megabytes).

MEM_RSS_MV = '%(rssM)'
RSS memory usage (in megabytes).

SIZE_PACKET_UWSGI = '%(pktsize)'
Size of the internal request uwsgi packet.

MOD1 = '%(modifier1)'
`modifier1` of the request. See `.routing.modifiers`.

MOD2 = '%(modifier2)'
`modifier2` of the request. See `.routing.modifiers`.

metric
Metric value (see The Metrics subsystem).

alias of *VarMetric*

request_var
Request variable value.

alias of *VarRequestVar*

3.8.8 Main process

Actions

class uwsgiconf.options.main_process_actions.**HookAction**(**args*)

class uwsgiconf.options.main_process_actions.**ActionMount**(*mountpoint*, ***, *fs=None*, *src=None*, *flags=None*)

Mount or unmount filesystems.
Examples:

- Mount: proc none /proc
- Unmount: /proc

Parameters

- **mountpoint** (*str*) –
- **fs** (*str*) – Filesystem. Presence indicates mounting.
- **src** (*str*) – Presence indicates mounting.
- **flags** (*str|list*) – Flags available for the operating system. As an example on Linux you will options like: bind, recursive, readonly, rec, detach etc.

class uwsgiconf.options.main_process_actions.**ActionExecute**(*command*)
Run the shell command.

Command run under `/bin/sh`. If for some reason you do not want to use `/bin/sh`, use `binsh` option,
Examples:

- cat /proc/self/mounts

class uwsgiconf.options.main_process_actions.**ActionCall**(*target*, *, *honour_exit_status=False*, *arg_int=False*)
Call functions in the current process address space.

Parameters

- **target** (*str*) – Symbol and args.
- **honour_exit_status** (*bool*) – Expect an int return. Anything != 0 means failure.
- **arg_int** (*bool*) – Parse the argument as an int.

class uwsgiconf.options.main_process_actions.**ActionDirChange**(*target_dir*)
Changes a directory.

Convenience action, same as `call:chdir <directory>`.

class uwsgiconf.options.main_process_actions.**ActionDirCreate**(*target_dir*)
Creates a directory with 0777.

class uwsgiconf.options.main_process_actions.**ActionFileCreate**(*fpath: Union[str, pathlib.Path]*)
Creates a directory with 0666.

class uwsgiconf.options.main_process_actions.**ActionExit**(*status_code=None*)
Exits.

Convenience action, same as `callint:exit [num]`.

class uwsgiconf.options.main_process_actions.**ActionPrintout**(*text=None*)
Prints.

Convenience action, same as calling the `uwsgi_log` symbol.

class uwsgiconf.options.main_process_actions.**ActionSetHostName**(*name*)
Sets a host name.

class uwsgiconf.options.main_process_actions.**ActionAlarm**(*alarm*, *message*)
Issues an alarm. See `.alarms` options group.

class uwsgiconf.options.main_process_actions.**ActionFileWrite**(*target*, *text*, *, *append=False*, *newline=False*)

Writes a string to the specified file.

If file doesn't exist it will be created.

Note: Since 1.9.21

Parameters

- **target** (*str*) – File to write to.
- **text** (*str*) – Text to write into file.
- **append** (*bool*) – Append text instead of rewrite.
- **newline** (*bool*) – Add a newline at the end.

class uwsgiconf.options.main_process_actions.**ActionFifoWrite**(*target*, *text*, *, *wait=False*)

Writes a string to the specified FIFO (see `fifo_file` from `master_process` params).

Parameters **wait** (*bool*) – Wait until FIFO is available.

class uwsgiconf.options.main_process_actions.**ActionUnlink**(*target*)

Unlink the specified file.

Note: Since 1.9.21

---

class uwsgiconf.options.main_process.**MainProcess**(**args*, ***kwargs*)

Main process is the uWSGI process.

> **Warning:** Do not run uWSGI instances as root. You can start your uWSGIs as root, but be sure to drop privileges with the `uid` and `gid` options from `set_owner_params`.

class actions

Actions available for `.set_hook()`.

alarm
alias of *uwsgiconf.options.main_process_actions.ActionAlarm*

call
alias of *uwsgiconf.options.main_process_actions.ActionCall*

dir_change
alias of *uwsgiconf.options.main_process_actions.ActionDirChange*

dir_create
alias of *uwsgiconf.options.main_process_actions.ActionDirCreate*

execute
alias of *uwsgiconf.options.main_process_actions.ActionExecute*

exit
alias of *uwsgiconf.options.main_process_actions.ActionExit*

fifo_write
alias of *uwsgiconf.options.main_process_actions.ActionFifoWrite*

file_create
alias of *uwsgiconf.options.main_process_actions.ActionFileCreate*

file_write
alias of *uwsgiconf.options.main_process_actions.ActionFileWrite*

mount
alias of *uwsgiconf.options.main_process_actions.ActionMount*

printout
alias of *uwsgiconf.options.main_process_actions.ActionPrintout*

set_host_name
alias of *uwsgiconf.options.main_process_actions.ActionSetHostName*

unlink
alias of *uwsgiconf.options.main_process_actions.ActionUnlink*

class phases
Phases available for hooking using `.set_hook()`.

Some of them may be **fatal** - a failing hook for them will mean failing of the whole uWSGI instance (generally calling exit(1)).

ASAP = 'asap'
As soon as possible. **Fatal**

Run directly after configuration file has been parsed, before anything else is done.

JAIL_PRE = 'pre-jail'
Before jailing. **Fatal**

Run before any attempt to drop privileges or put the process in some form of jail.

JAIL_IN = 'in-jail'
In jail after initialization. **Fatal**

Run soon after jayling, but after post-jail. If jailing requires fork(), the chidlren run this phase.

JAIL_POST = 'post-jail'
After jailing. **Fatal**

Run soon after any jailing, but before privileges drop. If jailing requires fork(), the parent process run this phase.

PRIV_DROP_PRE = 'as-root'
Before privileges drop. **Fatal**

Last chance to run something as root.

PRIV_DROP_POST = 'as-user'
After privileges drop. **Fatal**

MASTER_START = 'master-start'
When Master starts.

EMPEROR_START = 'emperor-start'
When Emperor starts.

EMPEROR_STOP = 'emperor-stop'
When Emperor sent a stop message.

EMPEROR_RELOAD = 'emperor-reload'
When Emperor sent a reload message.

EMPEROR_LOST = 'emperor-lost'
When Emperor connection is lost.

EXIT = 'as-user-atexit'
Before app exit and reload.

APP_LOAD_PRE = 'pre-app'
Before app loading. **Fatal**

APP_LOAD_POST = 'post-app'
After app loading. **Fatal**

VASSAL_ON_DEMAND_IN = 'as-on-demand-vassal'
Whenever a vassal enters on-demand mode.

VASSAL_CONFIG_CHANGE_POST = 'as-on-config-vassal'
Whenever the emperor detects a config change for an on-demand vassal.

VASSAL_START_PRE = 'as-emperor-before-vassal'
Before the new vassal is spawned.

VASSAL_PRIV_DRP_PRE = 'as-vassal-before-drop'
In vassal, before dropping its privileges.

VASSAL_SET_NAMESPACE = 'as-emperor-setns'
In the emperor entering vassal namespace.

VASSAL_START_IN = 'as-vassal'
In the vassal before executing the uwsgi binary. **Fatal**

In vassal on start just before calling exec() directly in the new namespace.

VASSAL_START_POST = 'as-emperor'
In the emperor soon after a vassal has been spawn.

Setting 4 env vars:

- UWSGI_VASSAL_CONFIG
- UWSGI_VASSAL_PID
- UWSGI_VASSAL_UID
- UWSGI_VASSAL_GID

GATEWAY_START_IN_EACH = 'as-gateway'
In each gateway on start.

MULE_START_IN_EACH = 'as-mule'
In each mule on start.

WORKER_ACCEPTING_PRE_EACH = 'accepting'
Before the each worker starts accepting requests.

Note: Since 1.9.21

WORKER_ACCEPTING_PRE_FIRST = 'accepting1'
Before the first worker starts accepting requests.

Note: Since 1.9.21

WORKER_ACCEPTING_PRE_EACH_ONCE = 'accepting-once'
Before the each worker starts accepting requests, one time per instance.

Note: Since 1.9.21

WORKER_ACCEPTING_PRE_FIRST_ONCE = 'accepting1-once'
Before the first worker starts accepting requests, one time per instance.

Note: Since 1.9.21

set_basic_params(*, *touch_reload: Union[str, List[str]] = None*, *priority: int = None*, *vacuum: bool = None*, *binary_path: str = None*, *honour_stdin: bool = None*)

Parameters

- **touch_reload** – Reload uWSGI if the specified file or directory is modified/touched.
- **priority** – Set processes/threads priority (`nice`) value.
- **vacuum** – Try to remove all of the generated files/sockets (UNIX sockets and pidfiles) upon exit.
- **binary_path** – Force uWSGI binary path. If you do not have uWSGI in the system path you can force its path with this option to permit the reloading system and the Emperor to easily find the binary to execute.
- **honour_stdin** – Do not remap stdin to `/dev/null`. By default, `stdin` is remapped to `/dev/null` on uWSGI startup. If you need a valid stdin (for debugging, piping and so on) use this option.

set_memory_params(*, *ksm_interval: int = None*, *no_swap: bool = None*)
Set memory related parameters.

Parameters

- **ksm_interval** – Kernel Samepage Merging frequency option, that can reduce memory usage. Accepts a number of requests (or master process cycles) to run page scanner after.

  Note: Linux only.

  - http://uwsgi.readthedocs.io/en/latest/KSM.html

- **no_swap** – Lock all memory pages avoiding swapping.

daemonize(*log_into: str*, *, *after_app_loading: bool = False*)
Daemonize uWSGI.

Parameters

- **log_into** (*str*) – Logging destination:
  - File: /tmp/mylog.log
  - UPD: 192.168.1.2:1717

> **Note:** This will require an UDP server to manage log messages. Use `networking.register_socket('192.168.1.2:1717, type=networking.SOCK_UDP)` to start uWSGI UDP server.

- **after_app_loading** – Whether to daemonize after or before applications loading.

change_dir(*to: str*, *, *after_app_loading: bool = False*)
Chdir to specified directory before or after apps loading.

Parameters

- **to** – Target directory.
- **after_app_loading** – *True* - after load *False* - before load

set_owner_params(*, *uid: Union[str, int] = None*, *gid: Union[str, int] = None*, *add_gids: Union[str, int, List[Union[str, int]]] = None*, *set_asap: bool = False*)
Set process owner params - user, group.

Parameters

- **uid** – Set uid to the specified username or uid.
- **gid** – Set gid to the specified groupname or gid.
- **add_gids** – Add the specified group id to the process credentials. This options allows you to add additional group ids to the current process. You can specify it multiple times.
- **set_asap** – Set as soon as possible. Setting them on top of your vassal file will force the instance to setuid()/setgid() as soon as possible and without the (theoretical) possibility to override them.

get_owner(*, *default: bool = True*) → Tuple[Union[str, int, None], Union[str, int, None]]
Return (User ID, Group ID) tuple

Parameters **default** – Whether to return default if not set.

set_hook(*phase: str, action: Union[str, HookAction, List[Union[str, HookAction]]]*)
Allows setting hooks (attaching actions) for various uWSGI phases.

Parameters

- **phase** – See constants in `.phases`.
- **action** –

set_hook_touch(*fpath: Union[str, pathlib.Path], action: Union[str, HookAction, List[Union[str, HookAction]]]*)
Allows running certain action when the specified file is touched.

Parameters

- **fpath** – File path.
- **action** –

set_hook_after_request(*func: str*)
Run the specified function/symbol (C level) after each request.

Parameters **func** –

set_on_exit_params(*, *skip_hooks: bool = None*, *skip_teardown: bool = None*)
Set params related to process exit procedure.

Parameters

- **skip_hooks** – Skip `EXIT` phase hook.

  Note: Ignored by the master.

- **skip_teardown** – Allows skipping teardown (finalization) processes for some plugins.

  Note: Ignored by the master.

  Supported by:

  - Perl
  - Python

run_command_on_event(*command: str*, *, *phase: str = 'asap'*)
Run the given command on a given phase.

Parameters

- **command** –
- **phase** – See constants in `Phases` class.

run_command_on_touch(*command: str*, *, *target: str*)
Run command when the specified file is modified/touched.

Parameters

- **command** –
- **target** – File path.

set_pid_file(*fpath: Union[str, pathlib.Path]*, *, *before_priv_drop: bool = True*, *safe: bool = False*)
Creates pidfile before or after privileges drop.

Parameters

- **fpath** – File path.
- **before_priv_drop** – Whether to create pidfile before privileges are dropped.

  Note: Vacuum is made after privileges drop, so it may not be able to delete PID file if it was created before dropping.

- **safe** – The safe-pidfile works similar to pidfile but performs the write a little later in the loading process. This avoids overwriting the value when app loading fails, with the consequent loss of a valid PID number.

set_naming_params(*, *autonaming: bool = None*, *prefix: str = None*, *suffix: str = None*, *name: str = None*)
Setups processes naming parameters.

Parameters

- **autonaming** – Automatically set process name to something meaningful. Generated process names may be 'uWSGI Master', 'uWSGI Worker #', etc.
- **prefix** – Add prefix to process names.
- **suffix** – Append string to process names.
- **name** – Set process names to given static value.

3.8.9 Master process

class uwsgiconf.options.master_process.**MasterProcess**(**args*, ***kwargs*)

Master process is a separate process offering mentoring capabilities for other processes. Only one master process per uWSGI instance.

uWSGI's built-in prefork+threading multi-worker management mode, activated by flicking the master switch on.

Note: For all practical serving deployments it's not really a good idea not to use master mode.

---

set_basic_params(**, enable: bool = None, name: str = None, no_orphans: bool = None, as_root: bool = None, subproc_check_interval: int = None, fifo_file: str = None*)

Parameters

- **enable** – Enable uWSGI master process.
- **name** – Set master process name to given value.
- **no_orphans** – Automatically kill workers if master dies (can be dangerous for availability).
- **as_root** – Leave master process running as root.
- **subproc_check_interval** – Set the interval (in seconds) of master checks. Default: 1 The master process makes a scan of subprocesses, etc. every N seconds.

  Warning: You can increase this time if you need to, but it's DISCOURAGED.

- **fifo_file** – Enables the master FIFO.

  Note: Placeholders can be used to build paths, e.g.: {project_runtime_dir}.fifo See `Section.project_name` and `Section.runtime_dir`.

  Instead of signals, you can tell the master to create a UNIX named pipe (FIFO) that you may use to issue commands to the master.

  Up to 10 different FIFO files supported. By default the first specified is bound (mapped as '0').

  – http://uwsgi.readthedocs.io/en/latest/MasterFIFO.html#the-master-fifo

  Note: Since 1.9.17

set_exit_events(*, *no_workers: bool = None*, *idle: bool = None*, *reload: bool = None*, *sig_term: bool = None*)

Do exit on certain events

Parameters

- **no_workers** (*bool*) – Shutdown uWSGI when no workers are running.
- **idle** (*bool*) – Shutdown uWSGI when idle.
- **reload** (*bool*) – Force exit even if a reload is requested.
- **sig_term** (*bool*) – Exit on SIGTERM instead of brutal workers reload.

Note: Before 2.1 SIGTERM reloaded the stack while SIGINT/SIGQUIT shut it down.

set_exception_handling_params(*, *handler: Union[str, List[str]] = None*, *catch: bool = None*, *no_write_exception: bool = None*)

Exception handling related params.

Parameters

- **handler** – Register one or more exception handling C-functions.
- **catch** – Catch exceptions and report them as http output (including stack trace and env params).

Warning: Use only for testing purposes.

- **no_write_exception** – Disable exception generation on write()/writev().

Note: This can be combined with `logging.set_filters(write_errors=False, sigpipe=False)`.

set_idle_params(*, *timeout: int = None*, *exit: bool = None*)

Activate idle mode - put uWSGI in cheap mode after inactivity timeout.

Parameters

- **timeout** – Inactivity timeout in seconds.
- **exit** – Shutdown uWSGI when idle.

set_reload_params(*, *mercy: int = None*, *exit: bool = None*)

Set reload related params.

Parameters

- **mercy** – Set the maximum time (in seconds) we wait for workers and other processes to die during reload/shutdown.
- **exit** – Force exit even if a reload is requested.

add_cron_task(*command: str*, *, *weekday: Union[str, int] = None*, *month: Union[str, int] = None*, *day: Union[str, int] = None*, *hour: Union[str, int] = None*, *minute: Union[str, int] = None*, *legion: str = None*, *unique: bool = None*, *harakiri: int = None*)

Adds a cron task running the given command on the given schedule. http://uwsgi.readthedocs.io/en/latest/Cron.html

HINTS:

- **Use negative values to say *every*:** hour=-3 stands for *every 3 hours*
- **Use - (minus) to make interval:** minute='13-18' stands for *from minute 13 to 18*

Note: We use cron2 option available since 1.9.11.

Parameters

- **command** – Command to execute on schedule (with or without path).
- **weekday** – Day of a the week number. Defaults to *each*. 0 - Sunday 1 - Monday 2 - Tuesday 3 - Wednesday 4 - Thursday 5 - Friday 6 - Saturday
- **month** – Month number 1-12. Defaults to *each*.
- **day** – Day of the month number 1-31. Defaults to *each*.
- **hour** – Hour 0-23. Defaults to *each*.
- **minute** – Minute 0-59. Defaults to *each*.
- **legion** – Set legion (cluster) name to use this cron command against. Such commands are only executed by legion lord node.
- **unique** – Marks command as unique. Default to not unique. Some commands can take a long time to finish or just hang doing their thing. Sometimes this is okay, but there are also cases when running multiple instances of the same command can be dangerous.
- **harakiri** – Enforce a time limit (in seconds) on executed commands. If a command is taking longer it will be killed.

attach_process_classic(*command_or_pid_path: str*, ***, *background: bool*, *control: bool = False*, *for_legion: bool = False*)

Attaches a command/daemon to the master process optionally managed by a pidfile.

This will allow the uWSGI master to control/monitor/respawn this process.

Note: This uses old classic uWSGI means of process attaching To have more control use `.attach_process()` method (requires uWSGI 2.0+)

---

http://uwsgi-docs.readthedocs.io/en/latest/AttachingDaemons.html

Parameters

- **command_or_pid_path** –
- **background** – Must indicate whether process is in background.
- **control** – Consider this process a control: when the daemon dies, the master exits.

  Note: pidfile managed processed not supported.

- **for_legion** – Legion daemons will be executed only on the legion lord node, so there will always be a single daemon instance running in each legion. Once the lord dies a daemon will be spawned on another node.

Note: uWSGI 1.9.9+ required.

attach_process(*command: str*, ***, *for_legion: bool = False*, *broken_counter: int = None*, *pidfile: str = None*, *control: bool = None*, *daemonize: bool = None*, *touch_reload: Union[str, List[str]] = None*, *signal_stop: int = None*, *signal_reload: int = None*, *honour_stdin: bool = None*, *uid: Union[str, int] = None*, *gid: Union[str, int] = None*, *new_pid_ns: bool = None*, *change_dir: str = None*)

Attaches a command/daemon to the master process.

This will allow the uWSGI master to control/monitor/respawn this process.

http://uwsgi-docs.readthedocs.io/en/latest/AttachingDaemons.html

Parameters

- **command** – The command line to execute.
- **for_legion** – Legion daemons will be executed only on the legion lord node, so there will always be a single daemon instance running in each legion. Once the lord dies a daemon will be spawned on another node.
- **broken_counter** – Maximum attempts before considering a daemon "broken".
- **pidfile** – The pidfile path to check (enable smart mode).
- **control** – If True, the daemon becomes a *control* one: if it dies the whole uWSGI instance dies.
- **daemonize** – Daemonize the process (enable smart2 mode).
- **touch_reload** – List of files to check: whenever they are 'touched', the daemon is restarted
- **signal_stop** – The signal number to send to the daemon when uWSGI is stopped.
- **signal_reload** – The signal number to send to the daemon when uWSGI is reloaded.
- **honour_stdin** – The signal number to send to the daemon when uWSGI is reloaded.
- **uid** – Drop privileges to the specified uid.

  Note: Requires master running as root.

- **gid** – Drop privileges to the specified gid.

  Note: Requires master running as root.

- **new_pid_ns** – Spawn the process in a new pid namespace.

  Note: Requires master running as root.

  Note: Linux only.

- **change_dir** – Use chdir() to the specified directory before running the command.

3.8.10 Monitoring

Metric Types

class uwsgiconf.options.monitoring_metric_types.**Metric**(*name*, *, *oid=None*, *alias_for=None*, *collector=None*, *initial_value=None*, *collect_interval=None*, *reset_after_push=None*)

Parameters

- **name** (*str*) – Metric name.

  Note: Only numbers, letters, underscores, dashes and dots.

- **alias_for** (*str*) – If set metric will be a simple alias for the specified one.
- **oid** (*str*) – Metric OID.

  Required for SNMP.

  - http://uwsgi-docs.readthedocs.io/en/latest/Metrics.html#oid-assigment-for-plugins
- **collector** (Collector) – Collector to be used. If not set it is considered that the value must be updated manually from applications using the metrics API.

  - http://uwsgi-docs.readthedocs.io/en/latest/Metrics.html#api
- **initial_value** (*int*) – Set the metric to a specific value on startup.
- **collect_interval** (*int*) – How ofter the metric should be gathered. In seconds.
- **reset_after_push** (*bool*) – Reset the metric to zero (or the configured initial_value) after it's been pushed.

class uwsgiconf.options.monitoring_metric_types.**MetricTypeCounter**(*name*, *, *oid=None*, *alias_for=None*, *collector=None*, *initial_value=None*, *collect_interval=None*, *reset_after_push=None*)

A generally-growing up number.
Example:

- number of requests

Parameters

- **name** (*str*) – Metric name.

  Note: Only numbers, letters, underscores, dashes and dots.

- **alias_for** (*str*) – If set metric will be a simple alias for the specified one.
- **oid** (*str*) – Metric OID.

  Required for SNMP.

  - http://uwsgi-docs.readthedocs.io/en/latest/Metrics.html#oid-assigment-for-plugins

- **collector** (Collector) – Collector to be used. If not set it is considered that the value must be updated manually from applications using the metrics API.

  - http://uwsgi-docs.readthedocs.io/en/latest/Metrics.html#api

- **initial_value** (*int*) – Set the metric to a specific value on startup.
- **collect_interval** (*int*) – How ofter the metric should be gathered. In seconds.
- **reset_after_push** (*bool*) – Reset the metric to zero (or the configured initial_value) after it's been pushed.

class uwsgiconf.options.monitoring_metric_types.**MetricTypeGauge**(*name*, ***, *oid=None*, *alias_for=None*, *collector=None*, *initial_value=None*, *collect_interval=None*, *reset_after_push=None*)

A number that can increase or decrease dynamically.
Example:

- memory used by a worker
- CPU load

Parameters

- **name** (*str*) – Metric name.

  Note: Only numbers, letters, underscores, dashes and dots.

- **alias_for** (*str*) – If set metric will be a simple alias for the specified one.
- **oid** (*str*) – Metric OID.

  Required for SNMP.

  - http://uwsgi-docs.readthedocs.io/en/latest/Metrics.html#oid-assigment-for-plugins

- **collector** (Collector) – Collector to be used. If not set it is considered that the value must be updated manually from applications using the metrics API.

  - http://uwsgi-docs.readthedocs.io/en/latest/Metrics.html#api

- **initial_value** (*int*) – Set the metric to a specific value on startup.
- **collect_interval** (*int*) – How ofter the metric should be gathered. In seconds.
- **reset_after_push** (*bool*) – Reset the metric to zero (or the configured initial_value) after it's been pushed.

class uwsgiconf.options.monitoring_metric_types.**MetricTypeAbsolute**(*name*, *, *oid=None*, *alias_for=None*, *collector=None*, *initial_value=None*, *collect_interval=None*, *reset_after_push=None*)

An absolute number.
Example:

- memory of the whole server
- size of the hard disk.

Parameters

- **name** (*str*) – Metric name.

  Note: Only numbers, letters, underscores, dashes and dots.

- **alias_for** (*str*) – If set metric will be a simple alias for the specified one.
- **oid** (*str*) – Metric OID.

  Required for SNMP.

  - http://uwsgi-docs.readthedocs.io/en/latest/Metrics.html#oid-assigment-for-plugins

- **collector** (Collector) – Collector to be used. If not set it is considered that the value must be updated manually from applications using the metrics API.

  - http://uwsgi-docs.readthedocs.io/en/latest/Metrics.html#api

- **initial_value** (*int*) – Set the metric to a specific value on startup.
- **collect_interval** (*int*) – How ofter the metric should be gathered. In seconds.
- **reset_after_push** (*bool*) – Reset the metric to zero (or the configured initial_value) after it's been pushed.

class uwsgiconf.options.monitoring_metric_types.**MetricTypeAlias**(*name*, *, *oid=None*, *alias_for=None*, *collector=None*, *initial_value=None*, *collect_interval=None*, *reset_after_push=None*)

This is a virtual metric pointing to another one .

You can use it to give different names to already existing metrics.

Parameters

- **name** (*str*) – Metric name.

  Note: Only numbers, letters, underscores, dashes and dots.

- **alias_for** (*str*) – If set metric will be a simple alias for the specified one.
- **oid** (*str*) – Metric OID.

  Required for SNMP.

  - http://uwsgi-docs.readthedocs.io/en/latest/Metrics.html#oid-assigment-for-plugins

- **collector** (Collector) – Collector to be used. If not set it is considered that the value must be updated manually from applications using the metrics API.

  - http://uwsgi-docs.readthedocs.io/en/latest/Metrics.html#api

- **initial_value** (*int*) – Set the metric to a specific value on startup.
- **collect_interval** (*int*) – How ofter the metric should be gathered. In seconds.
- **reset_after_push** (*bool*) – Reset the metric to zero (or the configured initial_value) after it's been pushed.

Pushers

class uwsgiconf.options.monitoring_pushers.**Pusher**(**args*)

class uwsgiconf.options.monitoring_pushers.**PusherSocket**(*address*, *, *prefix=None*)

Push metrics to a UDP server.

Uses the following format: `<metric> <type> <value>` `<type>` - is in the numeric form of metric type.

Parameters

- **address** (*str*) –
- **prefix** (*str*) – Arbitrary prefix to differentiate sender.

class uwsgiconf.options.monitoring_pushers.**PusherRrdtool**(*target_dir*, *, *library=None*, *push_interval=None*)

This will store an rrd file for each metric in the specified directory.

Each rrd file has a single data source named "metric".

Parameters

- **target_dir** (*str*) – Directory to store rrd files into.
- **library** (*str*) – Set the name of rrd library. Default: librrd.so.
- **push_interval** (*int*) – Set push frequency.

class uwsgiconf.options.monitoring_pushers.**PusherStatsd**(*address*, *, *prefix=None*, *no_workers=None*, *all_gauges=None*)

Push metrics to a statsd server.

Parameters

- **address** (*str*) –
- **prefix** (*str*) – Arbitrary prefix to differentiate sender.
- **no_workers** (*bool*) – Disable generation of single worker metrics.
- **all_gauges** (*bool*) – Push all metrics to statsd as gauges.

class uwsgiconf.options.monitoring_pushers.**PusherCarbon**(*address*, *, *node_realm=None*, *node_root=None*, *push_interval=None*, *idle_avg_source=None*, *use_metrics=None*, *no_workers=None*, *timeout=None*, *retries=None*, *retries_delay=None*, *hostname_dots_replacer=None*)

Push metrics to a Carbon server of Graphite.

Metric node format: `<node_root>.hostname.<node_realm>.metrics_data`.

- http://uwsgi.readthedocs.io/en/latest/Carbon.html
- http://uwsgi.readthedocs.io/en/latest/tutorials/GraphiteAndMetrics.html

Parameters

- **address** (*str|list[str]*) – Host and port. Example: 127.0.0.1:2004
- **node_realm** (*str*) – Set carbon metrics realm node.
- **node_root** (*str*) – Set carbon metrics root node. Default: uwsgi.
- **push_interval** (*int*) – Set carbon push frequency in seconds. Default: 60.
- **no_workers** (*bool*) – Disable generation of single worker metrics.
- **idle_avg_source** (*str*) – Average values source during idle period (no requests).

  Variants:

  - last (default)
  - zero
  - none

- **use_metrics** (*bool*) – Don't compute all statistics, use metrics subsystem data instead.

  Warning: Key names of built-in stats are different from those of metrics system.

- **timeout** (*int*) – Set carbon connection timeout in seconds. Default: 3.
- **retries** (*int*) – Set maximum number of retries in case of connection errors. Default: 1.
- **retries_delay** (*int*) – Set connection retry delay in seconds. Default: 7.
- **hostname_dots_replacer** (*str*) – Set char to use as a replacement for dots in hostname in *<node_root>.hostname.<node_realm>.metrics_data*‘

  This affects Graphite aggregation mechanics.

  Note: Dots are not replaced by default.

class uwsgiconf.options.monitoring_pushers.**PusherZabbix**(*address*, *, *prefix=None*, *template=None*)

Push metrics to a zabbix server.

Parameters

- **address** (*str*) –
- **prefix** (*str*) – Arbitrary prefix to differentiate sender.
- **template** (*str*) – Print (or store to a file) the zabbix template for the current metrics setup.

class uwsgiconf.options.monitoring_pushers.**PusherMongo**(*, *address=None*, *collection=None*, *push_interval=None*)

Push statistics (as JSON) the the specified MongoDB database.

Parameters

- **address** (*str*) – Default: 127.0.0.1:27017
- **collection** (*str*) – MongoDB colection to write into. Default: uwsgi.statistics
- **push_interval** (*int*) – Write interval in seconds.

class uwsgiconf.options.monitoring_pushers.**PusherFile**(*fpath=None*, *, *separator=None*, *push_interval=None*)

Stores stats JSON into a file.

Note: Mainly for demonstration purposes.

Parameters

- **fpath** (*str*) – File path. Default: uwsgi.stats
- **separator** (*str*) – New entry separator. Default:
- **push_interval** (*int*) – Write interval in seconds.

Collectors

class uwsgiconf.options.monitoring_collectors.**Collector**(**args*, ***kwargs*)

class uwsgiconf.options.monitoring_collectors.**CollectorPointer**(**args*, ***kwargs*)

The value is collected from memory pointer.

class uwsgiconf.options.monitoring_collectors.**CollectorFile**(*fpath*, *, *get_slot=None*)

The value is collected from a file.

Parameters

- **fpath** (*str*) – File path.
- **get_slot** (*int*) – Get value from the given slot number. Slots: the content is split (using n, t, spaces, r and zero as separator) and the item (the returned array is zero-based) used as the return value.

class uwsgiconf.options.monitoring_collectors.**CollectorFunction**(*func*)

The value is computed calling a specific C function every time.

Note:

- The argument it takes is a `uwsgi_metric` pointer. You generally do not need to parse the metric, so just casting to void will avoid headaches.
- The function must returns an `int64_t` value.

Parameters **func** (*str*) – Function to call.

class uwsgiconf.options.monitoring_collectors.**CollectorSum**(*what*)

The value is the sum of other metrics.

class uwsgiconf.options.monitoring_collectors.**CollectorAvg**(*what*)

The value is the algebraic average of the children.

Note: Since 1.9.20

---

class uwsgiconf.options.monitoring_collectors.**CollectorAccumulator**(*what*)

Always add the sum of children to the final value.

Example:

- Round 1: child1 = 22, child2 = 17 -> metric_value = 39
- Round 2: child1 = 26, child2 = 30 -> metric_value += 56

class uwsgiconf.options.monitoring_collectors.**CollectorAdder**(*what*, *value*)

Add the specified argument (arg1n) to the sum of children.

Parameters **value** (*int*) – Value to add (multiply if it is CollectorMultiplier).

class uwsgiconf.options.monitoring_collectors.**CollectorMultiplier**(*what*, *value*)

Multiply the sum of children by the specified argument.

Example:

- child1 = 22, child2 = 17, arg1n = 3 -> metric_value = (22+17)*3

Parameters **value** (*int*) – Value to add (multiply if it is CollectorMultiplier).

class uwsgiconf.options.monitoring.**Monitoring**(**args*, ***kwargs*)

Monitoring facilities.

- SNMP - http://uwsgi.readthedocs.io/en/latest/SNMP.html
- **Stats - http://uwsgi.readthedocs.io/en/latest/StatsServer.html** Set of metrics gathered from uWSGI internals.
- **Metrics - http://uwsgi.readthedocs.io/en/latest/Metrics.html** Basic set of metrics gathered from uWSGI internals + user defined metrics.

class metric_types

Various metric types to represent data of various nature.

User metrics must inherit from one of those.

absolute
alias of *uwsgiconf.options.monitoring_metric_types.MetricTypeAbsolute*

alias
alias of *uwsgiconf.options.monitoring_metric_types.MetricTypeAlias*

counter
alias of *uwsgiconf.options.monitoring_metric_types.MetricTypeCounter*

gauge
alias of *uwsgiconf.options.monitoring_metric_types.MetricTypeGauge*

class collectors
Metric collection and accumulation means.

accumulator
alias of *uwsgiconf.options.monitoring_collectors.CollectorAccumulator*

adder
alias of *uwsgiconf.options.monitoring_collectors.CollectorAdder*

avg
alias of *uwsgiconf.options.monitoring_collectors.CollectorAvg*

file
alias of *uwsgiconf.options.monitoring_collectors.CollectorFile*

function
alias of *uwsgiconf.options.monitoring_collectors.CollectorFunction*

multiplier
alias of *uwsgiconf.options.monitoring_collectors.CollectorMultiplier*

pointer
alias of *uwsgiconf.options.monitoring_collectors.CollectorPointer*

sum
alias of *uwsgiconf.options.monitoring_collectors.CollectorSum*

class pushers
Means to deliver metrics to various remotes or locals.

These are available for `.register_stats_pusher()`.

carbon
alias of *uwsgiconf.options.monitoring_pushers.PusherCarbon*

file
alias of *uwsgiconf.options.monitoring_pushers.PusherFile*

mongo
alias of *uwsgiconf.options.monitoring_pushers.PusherMongo*

rrdtool
alias of *uwsgiconf.options.monitoring_pushers.PusherRrdtool*

socket
alias of *uwsgiconf.options.monitoring_pushers.PusherSocket*

statsd
alias of *uwsgiconf.options.monitoring_pushers.PusherStatsd*

zabbix
alias of *uwsgiconf.options.monitoring_pushers.PusherZabbix*

register_metric(*metric*)
Officially Registered Metrics:

- **worker 3 - exports information about workers.** Example: worker.1.requests **[or 3.1.1]** reports the number of requests served by worker 1.
- **plugin 4 - namespace for metrics automatically added by plugins.** Example: plugins.foo.bar
- core 5 - namespace for general instance information.
- **router 6 - namespace for corerouters.** Example: router.http.active_sessions
- **socket 7 - namespace for sockets.** Example: socket.0.listen_queue
- **mule 8 - namespace for mules.** Example: mule.1.signals
- **spooler 9 - namespace for spoolers.** Example: spooler.1.signals
- system 10 - namespace for system metrics, like loadavg or free memory.

Parameters **metric** (*Metric|list[Metric]*) – Metric object.

set_metrics_params(*enable=None*, *store_dir=None*, *restore=None*, *no_cores=None*)
Sets basic Metrics subsystem params.

uWSGI metrics subsystem allows you to manage "numbers" from your apps.

When enabled, the subsystem configures a vast amount of metrics (like requests per-core, memory usage, etc) but, in addition to this, you can configure your own metrics, such as the number of active users or, say, hits of a particular URL, as well as the memory consumption of your app or the whole server.

- http://uwsgi.readthedocs.io/en/latest/Metrics.html
- SNMP Integration - http://uwsgi.readthedocs.io/en/latest/Metrics.html#snmp-integration

Parameters

- **enable** (*bool*) – Enables the subsystem.
- **store_dir** (*str*) – Directory to store metrics. The metrics subsystem can expose all of its metrics in the form of text files in a directory. The content of each file is the value of the metric (updated in real time).

  Note: Placeholders can be used to build paths, e.g.: {project_runtime_dir}/metrics/ See Section.project_name and Section.runtime_dir.

- **restore** (*bool*) – Restore previous metrics from store_dir. When you restart a uWSGI instance, all of its metrics are reset. Use the option to force the metric subsystem to read-back the values from the metric directory before starting to collect values.
- **no_cores** (*bool*) – Disable generation of cores-related metrics.

set_metrics_threshold(*name*, *value*, ***, *check_interval=None*, *reset_to=None*, *alarm=None*, *alarm_message=None*)
Sets metric threshold parameters.

Parameters

- **name** (*str*) – Metric name.
- **value** (*int*) – Threshold value.
- **reset_to** (*int*) – Reset value to when threshold is reached.
- **check_interval** (*int*) – Threshold check interval in seconds.
- **alarm** (*str|AlarmType*) – Alarm to trigger when threshold is reached.
- **alarm_message** (*str*) – Message to pass to alarm. If not set metrics name is passed.

set_stats_params (*address=None*, *enable_http=None*, *minify=None*, *no_cores=None*, *no_metrics=None*, *push_interval=None*)

Enables stats server on the specified address.

- http://uwsgi.readthedocs.io/en/latest/StatsServer.html

Parameters

- **address** (*str*) – Address/socket to make stats available on.

  Examples:

  - 127.0.0.1:1717
  - /tmp/statsock
  - :5050
- **enable_http** (*bool*) – Server stats over HTTP. Prefixes stats server json output with http headers.
- **minify** (*bool*) – Minify statistics json output.
- **no_cores** (*bool*) – Disable generation of cores-related stats.
- **no_metrics** (*bool*) – Do not include metrics in stats output.
- **push_interval** (*int*) – Set the default frequency of stats pushers in seconds/

register_stats_pusher (*pusher*)

Registers a pusher to be used for pushing statistics to various remotes/locals.

Parameters **pusher** (*Pusher|list[Pusher]*) –

enable_snmp (*address*, *community_string*)

Enables SNMP.

uWSGI server embeds a tiny SNMP server that you can use to integrate your web apps with your monitoring infrastructure.

- http://uwsgi.readthedocs.io/en/latest/SNMP.html

Note: SNMP server is started in the master process after dropping the privileges. If you want it to listen on a privileged port, you can either use Capabilities on Linux, or use the `as-root` option to run the master process as root.

Parameters

- **address** (*str*) – UDP address to bind to.

  Examples:

- 192.168.1.1:2222

- **community_string** (*str*) – SNMP instance identifier to address it.

3.8.11 Networking

Sockets

class uwsgiconf.options.networking_sockets.**Socket**(*address*, *, *bound_workers=None*, *modifier=None*)

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme. If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.
- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking_sockets.**SocketDefault**(*address*, *, *bound_workers=None*, *modifier=None*)

Bind using default protocol. See `default_socket_type` option.

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme. If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.
- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking_sockets.**SocketHttp**(*address*, *, *http11=False*, *bound_workers=None*, *modifier=None*)

Bind to the specified socket using HTTP

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **http11** (*bool*) – Keep-Alive support. If set the server will try to maintain the connection opened if a bunch of rules are respected.

  This is not a smart http 1.1 parser (to avoid parsing the whole response) but assumes the developer is generating the right headers.

  This has been added to support RTSP protocol for video streaming.
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme.

If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.

- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking_sockets.**SocketHttps**(*address*, *, *cert*, *key*, *ciphers=None*, *client_ca=None*, *bound_workers=None*, *modifier=None*)

Bind to the specified socket using HTTPS

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **cert** (*str*) – Certificate file.
- **key** (*str*) – Private key file.
- **ciphers** (*str*) – Ciphers [alias] string.

  Example:

  - DEFAULT
  - HIGH
  - DHE, EDH

  - https://www.openssl.org/docs/man1.1.0/apps/ciphers.html

- **client_ca** (*str*) – Client CA file for client-based auth.
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme. If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.
- **modifier** (*Modifier*) – Socket routing modifier.

classmethod get_certbot_paths(*domain: str*) → Tuple[str, str]

Returns a tuple of paths for files (certificates_chain, private_key) from Certbot https://certbot.eff.org

Those paths can be used to pass into Socket initializer.

Note: If files not found empty strings are returned.

Parameters **domain** – Domain name to get filepaths for.

class uwsgiconf.options.networking_sockets.**SocketUwsgi**(*address*, *, *persistent=False*, *bound_workers=None*, *modifier=None*)

uwSGI specific socket using `uwsgi` protocol.

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **persistent** (*bool*) – Use persistent uwsgi protocol (puwsgi).

- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme. If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.
- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking_sockets.**SocketUwsgis**(*address*, ***, *cert*, *key*, *ciphers=None*, *client_ca=None*, *bound_workers=None*, *modifier=None*)

uwSGI specific socket using `uwsgi` protocol over SSL.

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **cert** (*str*) – Certificate file.
- **key** (*str*) – Private key file.
- **ciphers** (*str*) – Ciphers [alias] string.

  Example:

  - DEFAULT
  - HIGH
  - DHE, EDH

  - https://www.openssl.org/docs/man1.1.0/apps/ciphers.html

- **client_ca** (*str*) – Client CA file for client-based auth.
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme. If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.
- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking_sockets.**SocketUdp**(*address*, ***, *bound_workers=None*, *modifier=None*)

Run the udp server on the specified address.

Note: Mainly useful for SNMP or shared UDP logging.

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme.

If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.

- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking_sockets.**SocketFastcgi**(*address*, *, *nph=False*, *bound_workers=None*, *modifier=None*)

Bind to the specified socket using FastCGI.

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **nph** (*bool*) – Use NPH mode ("no-parsed-header" - bypass the server completely by sending the complete HTTP header directly to the browser).
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme. If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.
- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking_sockets.**SocketScgi**(*address*, *, *nph=False*, *bound_workers=None*, *modifier=None*)

Bind to the specified UNIX/TCP socket using SCGI protocol.

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **nph** (*bool*) – Use NPH mode ("no-parsed-header" - bypass the server completely by sending the complete HTTP header directly to the browser).
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme. If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.
- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking_sockets.**SocketRaw**(*address*, *, *bound_workers=None*, *modifier=None*)

Bind to the specified UNIX/TCP socket using RAW protocol.

Raw mode allows you to directly parse the request in your application callable. Instead of getting a list of CGI vars/headers in your callable you only get the file descriptor soon after accept().

You can then read()/write() to that file descriptor in full freedom.

Note: Raw mode disables request logging.

Warning: Use it as a low-level socket wrapper.

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme. If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.
- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking_sockets.**SocketShared**(*address*, *, *undeferred=False*, *bound_workers=None*, *modifier=None*)

Create a shared socket for advanced jailing or IPC purposes.

Allows you to create a socket early in the server's startup and use it after privileges drop or jailing. This can be used to bind to privileged (<1024) ports.

Shared sockets are a way to share sockets among various uWSGI components: you can use that to share a socket between the fastrouter and uWSGI instance.

Parameters

- **address** (*str*) – Address ([host]:port or socket file) to bind socket to.
- **undeferred** (*bool*) – Use shared socket undeferred mode.
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme. If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.
- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking_sockets.**SocketZeromq**(*address*, *, *bound_workers=None*, *modifier=None*)

Introduce zeromq pub/sub pair.

Parameters

- **address** (*str|SocketShared*) – Address ([host]:port or socket file) to bind socket to.
- **bound_workers** (*str|int|list*) – Map socket to specific workers. As you can bind a uWSGI instance to multiple sockets, you can use this option to map specific workers to specific sockets to implement a sort of in-process Quality of Service scheme. If you host multiple apps in the same uWSGI instance, you can easily dedicate resources to each of them.
- **modifier** (*Modifier*) – Socket routing modifier.

class uwsgiconf.options.networking.**Networking**(**args*, ***kwargs*)

Networking related stuff. Socket definition, binding and tuning.

class sockets

Available socket types to use with `.register_socket()`.

default

alias of *uwsgiconf.options.networking_sockets.SocketDefault*

fastcgi
alias of *uwsgiconf.options.networking_sockets.SocketFastcgi*

http
alias of *uwsgiconf.options.networking_sockets.SocketHttp*

https
alias of *uwsgiconf.options.networking_sockets.SocketHttps*

raw
alias of *uwsgiconf.options.networking_sockets.SocketRaw*

scgi
alias of *uwsgiconf.options.networking_sockets.SocketScgi*

shared
alias of *uwsgiconf.options.networking_sockets.SocketShared*

udp
alias of *uwsgiconf.options.networking_sockets.SocketUdp*

uwsgi
alias of *uwsgiconf.options.networking_sockets.SocketUwsgi*

uwsgis
alias of *uwsgiconf.options.networking_sockets.SocketUwsgis*

zeromq
alias of *uwsgiconf.options.networking_sockets.SocketZeromq*

classmethod from_dsn(*dsn*, *, *allow_shared_sockets=None*) → uwsgiconf.options.networking_sockets.Socket
Constructs socket configuration object from DSN.

Note: This will also automatically use shared sockets to bind to priviledged ports when non root.

Parameters

- **dsn** (*str*) – Data source name, e.g: * http://127.0.0.1:8000 * https://127.0.0.1:443?cert=/here/there.crt&key=/that/my.key

  Note: Some schemas: fastcgi, http, https, raw, scgi, shared, udp, uwsgi, suwsgi, zeromq

- **allow_shared_sockets** (*bool*) – Allows using shared sockets to bind to priviledged ports. If not provided automatic mode is enabled: shared are allowed if current user is not root.

:rtype Socket

set_basic_params(*, *queue_size=None*, *freebind=None*, *default_socket_type=None*)

Parameters

- **queue_size** (*int*) – Also known as a backlog. Every socket has an associated queue where request will be put waiting for a process to became ready to accept them. When this queue is full, requests will be rejected.

Default: 100 (an average value chosen by the maximum value allowed by default by your kernel).

Note: The maximum value is system/kernel dependent. Before increasing it you may need to increase your kernel limit too.

- **freebind** (*bool*) – Put socket in freebind mode. Allows binding to non-existent network addresses.

Note: Linux only.

- **default_socket_type** (*str*) – Force the socket type as default. See `.socket_types`.

set_socket_params(*, *send_timeout=None*, *keep_alive=None*, *no_defer_accept=None*, *buffer_send=None*, *buffer_receive=None*)

Sets common socket params.

Parameters

- **send_timeout** (*int*) – Send (write) timeout in seconds.
- **keep_alive** (*bool*) – Enable TCP KEEPALIVEs.
- **no_defer_accept** (*bool*) – Disable deferred `accept()` on sockets by default (where available) uWSGI will defer the accept() of requests until some data is sent by the client (this is a security/performance measure). If you want to disable this feature for some reason, specify this option.
- **buffer_send** (*int*) – Set SO_SNDBUF (bytes).
- **buffer_receive** (*int*) – Set SO_RCVBUF (bytes).

set_unix_socket_params(*, *abstract=None*, *permissions=None*, *owner=None*, *umask=None*)

Sets Unix-socket related params.

Parameters

- **abstract** (*bool*) – Force UNIX socket into abstract mode (Linux only).
- **permissions** (*str*) – UNIX sockets are filesystem objects that obey UNIX permissions like any other filesystem object.

  You can set the UNIX sockets' permissions with this option if your webserver would otherwise have no access to the uWSGI socket. When used without a parameter, the permissions will be set to 666. Otherwise the specified chmod value will be used.
- **owner** (*str*) – Chown UNIX sockets.
- **umask** (*str*) – Set UNIX socket umask.

set_bsd_socket_params(*, *port_reuse=None*)

Sets BSD-sockets related params.

Parameters **port_reuse** (*bool*) – Enable REUSE_PORT flag on socket to allow multiple instances binding on the same address (BSD only).

register_socket(*socket*)

Registers the given socket(s) for further use.

Parameters **socket** (*Socket|list[Socket]*) – Socket type object. See `.sockets`.

set_ssl_params(*, *verbose_errors=None*, *sessions_cache=None*, *sessions_timeout=None*, *session_context=None*, *raw_options=None*, *dir_tmp=None*, *client_cert_var=None*)

Parameters

- **verbose_errors** (*bool*) – Be verbose about SSL errors.
- **sessions_cache** (*str|bool*) – Use uWSGI cache for ssl sessions storage.

  Accepts either bool or cache name string.

  – http://uwsgi.readthedocs.io/en/latest/SSLScaling.html

  Warning: Please be sure to configure cache before setting this.

- **sessions_timeout** (*int*) – Set SSL sessions timeout in seconds. Default: 300.
- **session_context** (*str*) – Session context identifying string. Can be set to static shared value to avoid session rejection.

  Default: a value built from the HTTP server address.

  – http://uwsgi.readthedocs.io/en/latest/SSLScaling.html#setup-2-synchronize-caches-of-different-https-routers

- **raw_options** (*int|list[int]*) – Set a raw ssl option by its numeric value.
- **dir_tmp** (*str*) – Store ssl-related temp files (e.g. pem data) in the specified directory.
- **client_cert_var** (*str*) – Export uWSGI variable `HTTPS_CC` containing the raw client certificate.

set_sni_params(*name: str*, *, *cert: str*, *key: str*, *ciphers: str = None*, *client_ca: str = None*, *wildcard: bool = False*)

Allows setting Server Name Identification (virtual hosting for SSL nodes) params.

- http://uwsgi.readthedocs.io/en/latest/SNI.html

Parameters

- **name** – Node/server/host name.
- **cert** – Certificate file.
- **key** – Private key file.
- **ciphers** – Ciphers [alias] string.

  Example:

  – DEFAULT

  – HIGH

  – DHE, EDH

  – https://www.openssl.org/docs/man1.1.0/apps/ciphers.html

- **client_ca** – Client CA file for client-based auth.

- **wildcard** – Allow regular expressions in `name` (used for wildcard certificates).

set_sni_dir_params(*dir*, *ciphers=None*)
Enable checking for cert/key/client_ca file in the specified directory and create a sni/ssl context on demand.

Expected filenames:

- <sni-name>.crt
- <sni-name>.key
- <sni-name>.ca - this file is optional

- http://uwsgi.readthedocs.io/en/latest/SNI.html#massive-sni-hosting

Parameters

- **dir** (*str*) –
- **ciphers** (*str*) – Ciphers [alias] string.

  Example:

  - DEFAULT
  - HIGH
  - DHE, EDH

  - https://www.openssl.org/docs/man1.1.0/apps/ciphers.html

3.8.12 Queue

class uwsgiconf.options.queue.**Queue**(**args*, ***kwargs*)
Queue.

At the low level it is a simple block-based shared array, with two optional counters, one for stack-style, LIFO usage, the other one for FIFO.

http://uwsgi-docs.readthedocs.io/en/latest/Queue.html

enable(*size*, *, *block_size=None*, *store=None*, *store_sync_interval=None*)
Enables shared queue of the given size.

Parameters

- **size** (*int*) – Queue size.
- **block_size** (*int*) – Block size in bytes. Default: 8 KiB.
- **store** (*str*) – Persist the queue into file.
- **store_sync_interval** (*int*) – Store sync interval in master cycles (usually seconds).

3.8.13 Routing

Actions

class uwsgiconf.options.routing_actions.**RouteAction**(**args*)

class uwsgiconf.options.routing_actions.**ActionFlush**
Send the current contents of the transformation buffer to the client (without clearing the buffer).
- http://uwsgi.readthedocs.io/en/latest/Transformations.html#flushing-magic

class uwsgiconf.options.routing_actions.**ActionGzip**
Encodes the response buffer to gzip.

class uwsgiconf.options.routing_actions.**ActionToFile**(*filename*, *, *mode=None*)
Used for caching a response buffer into a static file.

class uwsgiconf.options.routing_actions.**ActionUpper**
Transforms each character in uppercase.

Mainly as an example of transformation plugin.

class uwsgiconf.options.routing_actions.**ActionChunked**
Encodes the output in HTTP chunked.

class uwsgiconf.options.routing_actions.**ActionTemplate**
Allows using a template file to expose everything from internal routing system into it.
- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.19.html#the-template-transformation

class uwsgiconf.options.routing_actions.**ActionFixContentLen**(*, *add_header=False*)
Fixes Content-length header.
Parameters **add_header** (*bool*) – Force header add instead of plain fix of existing header.

class uwsgiconf.options.routing_actions.**ActionDoContinue**
Stop scanning the internal routing table and continue to the selected request handler.

class uwsgiconf.options.routing_actions.**ActionDoBreak**(*code*, *, *return_body=False*)
Stop scanning the internal routing table and close the request.
Parameters
- **code** (*int*) – HTTP code
- **return_body** – Uses uWSGI's built-in status code and returns both status code and message body.

class uwsgiconf.options.routing_actions.**ActionLog**(*message*)
Print the specified message in the logs or do not log a request is `message` is `None`.
Parameters **message** (*str|None*) – Message to add into log. If `None` logging will be disabled for this request.

class uwsgiconf.options.routing_actions.**ActionOffloadOff**
Do not use offloading.

class uwsgiconf.options.routing_actions.**ActionAddVarLog**(*name*, *val*)
Add the specified logvar.
Parameters
- **name** (*str*) – Variable name.
- **val** – Variable value.

class uwsgiconf.options.routing_actions.**ActionDoGoto**(*where*)
Make a forward jump to the specified label or rule position.
Parameters **where** (*str|int*) – Rule number of label to go to.

class uwsgiconf.options.routing_actions.**ActionAddVarCgi**(*name*, *val*)
Add the specified CGI (environment) variable to the request.
Parameters
- **name** (*str*) – Variable name.

- **val** – Variable value.

class uwsgiconf.options.routing_actions.**ActionHeaderAdd**(*name*, *val*)
Add the specified HTTP header to the response.

Parameters

- **name** (*str*) – Header name.
- **val** – Header value.

class uwsgiconf.options.routing_actions.**ActionHeaderRemove**(*name*)
Remove the specified HTTP header from the response.

Parameters **name** (*str*) – Header name.

class uwsgiconf.options.routing_actions.**ActionHeadersOff**
Disable headers.

class uwsgiconf.options.routing_actions.**ActionHeadersReset**(*code*)
Clear the response headers, setting a new HTTP status code, useful for resetting a response.

Parameters **code** (*int*) – HTTP code.

class uwsgiconf.options.routing_actions.**ActionSignal**(*num*)
Raise the specified uwsgi signal.

Parameters **num** (*int*) – Signal number.

class uwsgiconf.options.routing_actions.**ActionSend**(*data*, *, *crnl: bool = False*)
Extremely advanced (and dangerous) function allowing you to add raw data to the response.

Parameters

- **data** – Data to add to response.
- **crnl** – Add carriage return and new line.

class uwsgiconf.options.routing_actions.**ActionRedirect**(*url*, *, *permanent=False*)
Return a HTTP 301/302 Redirect to the specified URL.

Parameters

- **url** (*str*) – URL to redirect to.
- **permanent** (*bool*) – If `True` use 301, otherwise 302.

class uwsgiconf.options.routing_actions.**ActionRewrite**(*rule*, *, *do_continue=False*)
A rewriting engine inspired by Apache mod_rewrite.

Rebuild PATH_INFO and QUERY_STRING according to the specified rules before the request is dispatched to the request handler.

Parameters

- **rule** (*str*) – A rewrite rule.
- **do_continue** (*bool*) – Stop request processing and continue to the selected request handler.

class uwsgiconf.options.routing_actions.**ActionRouteUwsgi**(*external_address=''*, *, *modifier=''*, *app=''*)
Rewrite the modifier1, modifier2 and optionally UWSGI_APPID values of a request or route the request to an external uwsgi server.

Parameters

- **external_address** (*str*) – External uWSGI server address (host:port).
- **modifier** (*Modifier*) – Set request modifier.
- **app** (*str*) – Set `UWSGI_APPID`.

class uwsgiconf.options.routing_actions.**ActionRouteExternal**(*address*, ***, *host_header=None*)

Route the request to an external HTTP server.

Parameters

- **address** (*str*) – External HTTP address (host:port)
- **host_header** (*str*) – HOST header value.

class uwsgiconf.options.routing_actions.**ActionAlarm**(*name*, *message*)

Triggers an alarm.

- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.6.html#the-alarm-routing-action

Parameters

- **name** (*str*) – Alarm name
- **message** (*str*) – Message to pass into alarm.

class uwsgiconf.options.routing_actions.**ActionServeStatic**(*fpath: Union[str, pathlib.Path]*)

Serve a static file from the specified physical path.

Parameters **fpath** – Static file path.

class uwsgiconf.options.routing_actions.**ActionAuthBasic**(*realm*, ***, *user=None*, *password=None*, *do_next=False*)

Use Basic HTTP Auth.

Parameters

- **realm** (*str*) –
- **user** (*str*) –
- **password** (*str*) – Password or htpasswd-like file.
- **do_next** (*bool*) – Allow next rule.

class uwsgiconf.options.routing_actions.**AuthLdap**(*realm*, *address*, ***, *base_dn=None*, *bind_dn=None*, *bind_password=None*, *filter=None*, *login_attr=None*, *log_level=None*, *do_next=False*)

Use Basic HTTP Auth.

Parameters

- **realm** (*str*) –
- **address** (*str*) – LDAP server URI
- **base_dn** (*str*) – Base DN used when searching for users.
- **bind_dn** (*str*) – DN used for binding. Required if the LDAP server does not allow anonymous searches.
- **bind_password** (*str*) – Password for the `bind_dn` user.
- **filter** (*str*) – Filter used when searching for users. Default: `(objectClass=*)`
- **login_attr** (*str*) – LDAP attribute that holds user login. Default: `uid`.
- **log_level** (*str*) – Log level.

  Supported values:

  - 0 - don't log any binds

- 1 - log authentication errors,
- 2 - log both successful and failed binds

- **do_next** (*bool*) – Allow next rule.

class uwsgiconf.options.routing_actions.**ActionSetHarakiri**(*timeout*)
Set harakiri timeout for the current request.

Parameters **timeout** (*int*) –

class uwsgiconf.options.routing_actions.**ActionDirChange**(*dir*)
Changes a directory.

Parameters **dir** (*str*) – Directory to change into.

class uwsgiconf.options.routing_actions.**ActionSetVarUwsgiAppid**(*app*)
Set UWSGI_APPID.

Bypass `SCRIPT_NAME` and `VirtualHosting` to let the user choose the mountpoint without limitations (or headaches).

The concept is very generic: `UWSGI_APPID` is the identifier of an application. If it is not found in the internal list of apps, it will be loaded.

Parameters **app** (*str*) – Application ID.

class uwsgiconf.options.routing_actions.**ActionSetVarRemoteUser**(*user*)
Set REMOTE_USER

Parameters **user** (*str*) – Username.

class uwsgiconf.options.routing_actions.**ActionSetVarUwsgiHome**(*dir*)
Set UWSGI_HOME

Parameters **dir** (*str*) – Directory to make a new home.

class uwsgiconf.options.routing_actions.**ActionSetVarUwsgiScheme**(*value*)
Set UWSGI_SCHEME.

Set the URL scheme when it cannot be reliably determined. This may be used to force HTTPS (with the value `https`), for instance.

- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.6.html#configuring-dynamic-apps-with-internal-routing

Parameters **value** (*str*) –

class uwsgiconf.options.routing_actions.**ActionSetVarScriptName**(*name*)
Set SCRIPT_NAME

Parameters **name** (*str*) – Script name

class uwsgiconf.options.routing_actions.**ActionSetVarRequestMethod**(*name*)
Set REQUEST_METHOD

Parameters **name** (*str*) – Method name.

class uwsgiconf.options.routing_actions.**ActionSetVarRequestUri**(*value*)
Set REQUEST_URI

Parameters **value** (*str*) – URI

class uwsgiconf.options.routing_actions.**ActionSetVarRemoteAddr**(*value*)
Set REMOTE_ADDR

Parameters **value** (*str*) – Address.

class uwsgiconf.options.routing_actions.**ActionSetVarPathInfo**(*value*)
Set PATH_INFO

Parameters **value** (*str*) – New info.

class uwsgiconf.options.routing_actions.**ActionSetVarDocumentRoot**(*value*)
Set DOCUMENT_ROOT

Parameters **value** (*str*) –

class uwsgiconf.options.routing_actions.**ActionSetUwsgiProcessName**(*name*)
Set uWSGI process name.

Parameters **name** (*str*) – New process name.

class uwsgiconf.options.routing_actions.**ActionFixVarPathInfo**
Fixes PATH_INFO taking into account script name.

This action allows you to set SCRIPT_NAME in nginx without bothering to rewrite the PATH_INFO (something nginx cannot afford).

- http://uwsgi.readthedocs.io/en/latest/Changelog-2.0.11.html#fixpathinfo-routing-action

class uwsgiconf.options.routing_actions.**ActionSetScriptFile**(*fpath: Union[str, pathlib.Path]*)
Set script file.

- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.6.html#configuring-dynamic-apps-with-internal-routing

Parameters **fpath** (*str*) – File path.

Dedicated routers

class uwsgiconf.options.routing_routers.**RouterBase**(*on=None*)

Parameters **on** (*SocketShared|str*) – Activates the router on the given address.

class uwsgiconf.options.routing_routers.**Forwarder**(**args*)

class uwsgiconf.options.routing_routers.**ForwarderPath**(*sockets_dir*)
Use the specified base (allows %s pattern) for mapping requests to UNIX sockets.
Examples:

- /tmp/sockets/
- /tmp/sockets/%s/uwsgi.sock

- http://uwsgi.readthedocs.io/en/latest/Fastrouter.html#way-1-fastrouter-use-base
- http://uwsgi.readthedocs.io/en/latest/Fastrouter.html#way-2-fastrouter-use-pattern

Parameters **sockets_dir** (*str*) – UNIX sockets directory. Allows %s to denote key (domain).

class uwsgiconf.options.routing_routers.**ForwarderCode**(*script*, *func*, ***, *modifier=None*)
Forwards requests to nodes returned by a function.

This allows using user defined functions to calculate. Function must accept key (domain).

- http://uwsgi.readthedocs.io/en/latest/Fastrouter.html#way-5-fastrouter-use-code-string

> **Warning:** Remember to not put blocking code in your functions. The router is totally non-blocking, do not ruin it!

Parameters

- **script** (*str*) – Script (module for Python) name to get function from.
- **func** (*str*) – Function name.
- **modifier** (*Modifier*) – Routing modifier.

class uwsgiconf.options.routing_routers.**ForwarderCache**(*cache_name=None*)
Uses uWSGI cache to get target nodes from.

- http://uwsgi.readthedocs.io/en/latest/Fastrouter.htmlway-3-fastrouter-use-cache

Parameters **cache_name** (*str*) – Cache name to use.

class uwsgiconf.options.routing_routers.**ForwarderSocket**(*socket*)
Forwards request to the specified uwsgi socket.

Parameters **socket** (*str*) – Socket filepath.

class uwsgiconf.options.routing_routers.**ForwarderSubscriptionServer**(*address*)
Forwards requests to nodes returned by the subscription server.

Subscriptions are simple UDP packets that instruct the router which domain maps to which instance or instances.

To subscribe to such a subscription server use *.subscriptions.subscribe()*.

- http://uwsgi.readthedocs.io/en/latest/Fastrouter.html#way-4-fastrouter-subscription-server

Parameters **address** (*str*) – Address (including port) to run the subscription server on.

class uwsgiconf.options.routing_routers.**RouterHttp**(*on=None*, **, forward_to=None*)
uWSGI includes an HTTP router/proxy/load-balancer that can forward requests to uWSGI workers.

The server can be used in two ways:

- embedded - automatically spawn workers and setup the communication socket
- standalone - you have to specify the address of a uwsgi socket to connect to

See *subscribe_to* argument to *.set_basic_params()*

Note: If you want to go massive (virtualhosting and zero-conf scaling) combine the HTTP router with the uWSGI Subscription Server.

Activates the router on the given address.

Parameters

- **on** (*SocketShared|str*) – Activates the router on the given address.
- **forward_to** (*Forwarder|str|list[str]*) – Where to forward requests. Expects a forwarder instance or one or more node names.

set_basic_params(***, *workers=None*, *zerg_server=None*, *fallback_node=None*, *concurrent_events=None*, *cheap_mode=None*, *stats_server=None*, *quiet=None*, *buffer_size=None*, *keepalive=None*, *resubscribe_addresses=None*)

Parameters

- **workers** (*int*) – Number of worker processes to spawn.
- **zerg_server** (*str*) – Attach the router to a zerg server.
- **fallback_node** (*str*) – Fallback to the specified node in case of error.
- **concurrent_events** (*int*) – Set the maximum number of concurrent events router can manage.

  Default: system dependent.
- **cheap_mode** (*bool*) – Enables cheap mode. When the router is in cheap mode, it will not respond to requests until a node is available. This means that when there are no nodes subscribed, only your local app (if any) will respond. When all of the nodes go down, the router will return in cheap mode.
- **stats_server** (*str*) – Router stats server address to run at.
- **quiet** (*bool*) – Do not report failed connections to instances.
- **buffer_size** (*int*) – Set internal buffer size in bytes. Default: page size.

- **keepalive** (*int*) – Allows holding the connection open even if the request has a body.
  - http://uwsgi.readthedocs.io/en/latest/HTTP.html#http-keep-alive

  Note: See http11 socket type for an alternative.

- **resubscribe_addresses** (*str|list[str]*) – Forward subscriptions to the specified subscription server.

set_connections_params(*, *harakiri=None*, *timeout_socket=None*, *retry_delay=None*, *timeout_headers=None*, *timeout_backend=None*)

Sets connection-related parameters.

Parameters

- **harakiri** (*int*) – Set gateway harakiri timeout (seconds).
- **timeout_socket** (*int*) – Node socket timeout (seconds). Used to set the SPDY timeout. This is the maximum amount of inactivity after the SPDY connection is closed.

  Default: 60.
- **retry_delay** (*int*) – Retry connections to dead static nodes after the specified amount of seconds. Default: 30.
- **timeout_headers** (*int*) – Defines the timeout (seconds) while waiting for http headers.

  Default: *socket_timeout*.
- **timeout_backend** (*int*) – Defines the timeout (seconds) when connecting to backend instances.

  Default: *socket_timeout*.

set_manage_params(*, *chunked_input=None*, *chunked_output=None*, *gzip=None*, *websockets=None*, *source_method=None*, *rtsp=None*, *proxy_protocol=None*)

Allows enabling various automatic management mechanics.

- http://uwsgi.readthedocs.io/en/latest/Changelog-1.9.html#http-router-keepalive-auto-chunking-auto-gzip-and-transp

Parameters

- **chunked_input** (*bool*) – Automatically detect chunked input requests and put the session in raw mode.
- **chunked_output** (*bool*) – Automatically transform output to chunked encoding during HTTP 1.1 keepalive (if needed).
- **gzip** (*bool*) – Automatically gzip content if uWSGI-Encoding header is set to gzip, but content size (Content-Length/Transfer-Encoding) and Content-Encoding are not specified.
- **websockets** (*bool*) – Automatically detect websockets connections and put the session in raw mode.
- **source_method** (*bool*) – Automatically put the session in raw mode for *SOURCE* HTTP method.
  - http://uwsgi.readthedocs.io/en/latest/Changelog-2.0.5.html#icecast2-protocol-helpers

- **rtsp** (*bool*) – Allow the HTTP router to detect RTSP and chunked requests automatically.
- **proxy_protocol** (*bool*) – Allows the HTTP router to manage PROXY1 protocol requests, such as those made by Haproxy or Amazon Elastic Load Balancer (ELB).

set_owner_params(*uid=None*, *gid=None*)

Drop http router privileges to specified user and group.

Parameters

- **uid** (*str|int*) – Set uid to the specified username or uid.
- **gid** (*str|int*) – Set gid to the specified groupname or gid.

class uwsgiconf.options.routing_routers.**RouterHttps**(*on*, ***, *cert*, *key*, *ciphers=None*, *client_ca=None*, *session_context=None*, *use_spdy=None*, *export_cert_var=None*)

uWSGI includes an HTTPS router/proxy/load-balancer that can forward requests to uWSGI workers.

The server can be used in two ways:

- embedded - automatically spawn workers and setup the communication socket
- standalone - you have to specify the address of a uwsgi socket to connect to

See *subscribe_to* argument to *.set_basic_params()*

Note: If you want to go massive (virtualhosting and zero-conf scaling) combine the HTTP router with the uWSGI Subscription Server.

Binds https router to run on the given address.

Parameters

- **on** (*SocketShared|str*) – Activates the router on the given address.
- **cert** (*str*) – Certificate file.
- **key** (*str*) – Private key file.
- **ciphers** (*str*) – Ciphers [alias] string.

  Example:

  - DEFAULT
  - HIGH
  - DHE, EDH

  – https://www.openssl.org/docs/man1.1.0/apps/ciphers.html

- **client_ca** (*str*) – Client CA file for client-based auth.
- **session_context** (*str*) – Session context identifying string. Can be set to static shared value to avoid session rejection.

  Default: a value built from the HTTP server address.

  – http://uwsgi.readthedocs.io/en/latest/SSLScaling.html#setup-2-synchronize-caches-of-different-https-routers

- **use_spdy** (*bool*) – Use SPDY.
- **export_cert_var** (*bool*) – Export uwsgi variable *HTTPS_CC* containing the raw client certificate.

class uwsgiconf.options.routing_routers.**RouterSsl**(*on*, *cert*, *key*, *forward_to=None*, *ciphers=None*, *client_ca=None*, *session_context=None*, *use_sni=None*)

Works in the same way as the RouterRaw, but will terminate ssl connections.

Supports SNI for implementing virtual hosting.

Activates the router on the given address.

Parameters

- **on** (*SocketShared|str*) – Activates the router on the given address.
- **cert** (*str*) – Certificate file.
- **key** (*str*) – Private key file.
- **forward_to** (*Forwarder|str|list[str]*) – Where to forward requests. Expects a forwarder instance or one or more node names.
- **ciphers** (*str*) – Ciphers [alias] string.

  Example:

  - DEFAULT
  - HIGH
  - DHE, EDH

  - https://www.openssl.org/docs/man1.1.0/apps/ciphers.html

- **client_ca** (*str*) – Client CA file for client-based auth.
- **session_context** (*str*) – Session context identifying string. Can be set to static shared value to avoid session rejection.

  Default: a value built from the HTTP server address.

  - http://uwsgi.readthedocs.io/en/latest/SSLScaling.html#setup-2-synchronize-caches-of-different-https-routers

- **use_sni** (*bool*) – Use SNI to route requests.

set_connections_params(*harakiri=None*, *timeout_socket=None*, *retry_delay=None*, *retry_max=None*)

Sets connection-related parameters.

Parameters

- **harakiri** (*int*) – Set gateway harakiri timeout (seconds).
- **timeout_socket** (*int*) – Node socket timeout (seconds). Default: 60.
- **retry_delay** (*int*) – Retry connections to dead static nodes after the specified amount of seconds. Default: 30.
- **retry_max** (*int*) – Maximum number of retries/fallbacks to other nodes. Default: 3.

class uwsgiconf.options.routing_routers.**RouterFast**(*on=None*, *, *forward_to=None*)

A proxy/load-balancer/router speaking the uwsgi protocol.

You can put it between your webserver and real uWSGI instances to have more control over the routing of HTTP requests to your application servers.

Activates the router on the given address.

Parameters

- **on** (*SocketShared|str*) – Activates the router on the given address.
- **forward_to** (*Forwarder|str|list[str]*) – Where to forward requests. Expects a forwarder instance or one or more node names.

set_basic_params(*, *workers=None*, *zerg_server=None*, *fallback_node=None*, *concurrent_events=None*, *cheap_mode=None*, *stats_server=None*, *quiet=None*, *buffer_size=None*, *fallback_nokey=None*, *subscription_key=None*, *emperor_command_socket=None*)

Parameters

- **workers** (*int*) – Number of worker processes to spawn.
- **zerg_server** (*str*) – Attach the router to a zerg server.
- **fallback_node** (*str*) – Fallback to the specified node in case of error.
- **concurrent_events** (*int*) – Set the maximum number of concurrent events router can manage.

  Default: system dependent.
- **cheap_mode** (*bool*) – Enables cheap mode. When the router is in cheap mode, it will not respond to requests until a node is available. This means that when there are no nodes subscribed, only your local app (if any) will respond. When all of the nodes go down, the router will return in cheap mode.
- **stats_server** (*str*) – Router stats server address to run at.
- **quiet** (*bool*) – Do not report failed connections to instances.
- **buffer_size** (*int*) – Set internal buffer size in bytes. Default: page size.
- **fallback_nokey** (*bool*) – Move to fallback node even if a subscription key is not found.
- **subscription_key** (*str*) – Skip uwsgi parsing and directly set a key.
- **emperor_command_socket** (*str*) – Set the emperor command socket that will receive spawn commands.

  See *.empire.set_emperor_command_params()*.

set_resubscription_params(*addresses=None*, *bind_to=None*)

You can specify a dgram address (udp or unix) on which all of the subscriptions request will be forwarded to (obviously changing the node address to the router one).

The system could be useful to build 'federated' setup.

- http://uwsgi.readthedocs.io/en/latest/Changelog-2.0.1.html#resubscriptions

Parameters

- **addresses** (*str|list[str]*) – Forward subscriptions to the specified subscription server.

- **bind_to** (*str|list[str]*) – Bind to the specified address when re-subscribing.

set_connections_params(*harakiri=None*, *timeout_socket=None*, *retry_delay=None*, *retry_max=None*, *defer=None*)
Sets connection-related parameters.

Parameters

- **harakiri** (*int*) – Set gateway harakiri timeout (seconds).
- **timeout_socket** (*int*) – Node socket timeout (seconds). Default: 60.
- **retry_delay** (*int*) – Retry connections to dead static nodes after the specified amount of seconds. Default: 30.
- **retry_max** (*int*) – Maximum number of retries/fallbacks to other nodes. Default: 3
- **defer** (*int*) – Defer connection delay, seconds. Default: 5.

set_postbuffering_params(*size=None*, *store_dir=None*)
Sets buffering params.

Web-proxies like nginx are "buffered", so they wait til the whole request (and its body) has been read, and then it sends it to the backends.

Parameters

- **size** (*int*) – The size (in bytes) of the request body after which the body will be stored to disk (as a temporary file) instead of memory.
- **store_dir** (*str*) – Put buffered files to the specified directory. Default: TMPDIR, /tmp/

set_owner_params(*uid=None*, *gid=None*)
Drop http router privileges to specified user and group.

Parameters

- **uid** (*str|int*) – Set uid to the specified username or uid.
- **gid** (*str|int*) – Set gid to the specified groupname or gid.

class uwsgiconf.options.routing_routers.**RouterRaw**(*on=None*, ***, *forward_to=None*)
A pure-TCP load balancer.

Can be used to load balance between the various HTTPS routers.

Activates the router on the given address.

Parameters

- **on** (*SocketShared|str*) – Activates the router on the given address.
- **forward_to** (*Forwarder|str|list[str]*) – Where to forward requests. Expects a forwarder instance or one or more node names.

set_connections_params(*harakiri=None*, *timeout_socket=None*, *retry_delay=None*, *retry_max=None*, *use_xclient=None*)
Sets connection-related parameters.

Parameters

- **harakiri** (*int*) – Set gateway harakiri timeout (seconds).
- **timeout_socket** (*int*) – Node socket timeout (seconds). Default: 60.

- **retry_delay** (*int*) – Retry connections to dead static nodes after the specified amount of seconds. Default: 30.
- **retry_max** (*int*) – Maximum number of retries/fallbacks to other nodes. Default: 3.
- **use_xclient** (*bool*) – Use the xclient protocol to pass the client address.

class uwsgiconf.options.routing_routers.**RouterForkPty**(*on=None*, *undeferred=False*)

Allows allocation of pseudoterminals in jails.

Dealing with containers is now a common deployment pattern. One of the most annoying tasks when dealing with jails/namespaces is 'attaching' to already running instances. The forkpty router aims at simplifyng the process giving a pseudoterminal server to your uWSGI instances. A client connect to the socket exposed by the forkpty router and get a new pseudoterminal connected to a process (generally a shell, but can be whatever you want).

Note: To be used in cooperation with *pty* plugin.

Binds router to run on the given address.

Parameters

- **on** (*SocketShared|str*) – Activates the router on the given address.
- **undeferred** (*bool*) – Run router in undeferred mode.

set_basic_params(*, *workers=None*, *zerg_server=None*, *fallback_node=None*, *concurrent_events=None*, *cheap_mode=None*, *stats_server=None*, *run_command=None*)

Parameters

- **workers** (*int*) – Number of worker processes to spawn.
- **zerg_server** (*str*) – Attach the router to a zerg server.
- **fallback_node** (*str*) – Fallback to the specified node in case of error.
- **concurrent_events** (*int*) – Set the maximum number of concurrent events router can manage.

  Default: system dependent.
- **cheap_mode** (*bool*) – Enables cheap mode. When the router is in cheap mode, it will not respond to requests until a node is available. This means that when there are no nodes subscribed, only your local app (if any) will respond. When all of the nodes go down, the router will return in cheap mode.
- **stats_server** (*str*) – Router stats server address to run at.
- **run_command** (*str*) – Run the specified command on every connection. Default: /bin/sh.

set_connections_params(*, *harakiri=None*, *timeout_socket=None*)

Sets connection-related parameters.

Parameters

- **harakiri** (*int*) – Set gateway harakiri timeout (seconds).
- **timeout_socket** (*int*) – Node socket timeout (seconds). Default: 60.

set_window_params(*cols=None*, *rows=None*)

Sets pty window params.

Parameters

- **cols** (*int*) –
- **rows** (*int*) –

class uwsgiconf.options.routing_routers.**RouterTunTap**(*on=None*, ***, *device=None*, *stats_server=None*, *gateway=None*)

The tuntap router is a non-blocking highly optimized ip router translating from tuntap device to socket streams.

Allows full user-space networking in jails.

It is meant as a replacement for the currently available networking namespaces approaches. Compared to *veth* or *macvlan* it is really simple and allows total control over the routing subsystem (in addition to a simple customizable firewalling engine).

Generally you spawn the tuntap router in the Emperor instance. Vassals will run in new namespaces in which they create a tuntap device attached to the tuntap router. UNIX sockets are the only way to connect to the tuntap router after jailing.

Vassals should connect to tuntap device.

Passing params will create a router device.

Parameters

- **on** (*str*) – Socket file.
- **device** (*str*) – Device name.
- **stats_server** (*str*) – Router stats server address to run at.
- **gateway** (*str*) – Gateway address.

set_basic_params(***, *use_credentials=None*, *stats_server=None*)

Parameters

- **use_credentials** (*str*) – Enable check of SCM_CREDENTIALS for tuntap client/server.
- **stats_server** (*str*) – Router stats server address to run at.

register_route(*src*, *dst*, ***, *gateway*)

Adds a routing rule to the tuntap router.

Parameters

- **src** (*str*) – Source/mask.
- **dst** (*str*) – Destination/mask.
- **gateway** (*str*) – Gateway address.

device_connect(*socket*, ***, *device_name*)

Add a tuntap device to the instance.

To be used in a vassal.

Parameters

- **socket** (*str*) – Router socket.

  Example: */run/tuntap_router.socket*.

- **device_name** (*str*) – Device.

  Example: *uwsgi0*.

device_add_rule(*, *direction*, *action*, *src*, *dst*, *target=None*)
Adds a tuntap device rule.

To be used in a vassal.

Parameters

- **direction** (*str*) – Direction:
  - in
  - out.
- **action** (*str*) – Action:
  - allow
  - deny
  - route
  - gateway.
- **src** (*str*) – Source/mask.
- **dst** (*str*) – Destination/mask.
- **target** (*str*) – Depends on action.
  - Route / Gateway: Accept addr:port

add_firewall_rule(*, *direction*, *action*, *src=None*, *dst=None*)
Adds a firewall rule to the router.

The TunTap router includes a very simple firewall for governing vassal's traffic. The first matching rule stops the chain, if no rule applies, the policy is "allow".

Parameters

- **direction** (*str*) – Direction:
  - in
  - out
- **action** (*str*) – Action:
  - allow
  - deny
- **src** (*str*) – Source/mask.
- **dst** (*str*) – Destination/mask

Modifiers

class uwsgiconf.options.routing_modifiers.**ModifierWsgi**(*submod=None*)
Standard WSGI request followed by the HTTP request body.

class uwsgiconf.options.routing_modifiers.**ModifierPsgi**(*submod=None*)
Standard PSGI request followed by the HTTP request body.

class uwsgiconf.options.routing_modifiers.**ModifierLua**(*submod=None*)
Standard LUA/WSAPI request followed by the HTTP request body.

class uwsgiconf.options.routing_modifiers.**ModifierRack**(*submod=None*)
Standard RACK request followed by the HTTP request body.

class uwsgiconf.options.routing_modifiers.**ModifierJvm**(*submod=None*)
Standard JVM request for The JWSGI interface and The Clojure/Ring JVM request handler followed by the HTTP request body.

SUB_RING = 1
Use Clojure/Ring JVM request handler.

class uwsgiconf.options.routing_modifiers.**ModifierCgi**(*submod=None*)
Standard Running CGI scripts on uWSGI request followed by the HTTP request body.

class uwsgiconf.options.routing_modifiers.**ModifierManage**(*submod=None*)
Management interface request: setup flag specified by `modifier2`.

For a list of management flag look at `ManagementFlag`.

class uwsgiconf.options.routing_modifiers.**ModifierPhp**(*submod=None*)
Standard Running PHP scripts in uWSGI request followed by the HTTP request body.

class uwsgiconf.options.routing_modifiers.**ModifierMono**(*submod=None*)
Standard The Mono ASP.NET plugin request followed by the HTTP request body.

class uwsgiconf.options.routing_modifiers.**ModifierSpooler**(*submod=None*)
The uWSGI Spooler request, the block vars is converted to a dictionary/hash/table and passed to the spooler callable.

class uwsgiconf.options.routing_modifiers.**ModifierSymcall**(*submod=None*)
Direct call to C-like symbols.

class uwsgiconf.options.routing_modifiers.**ModifierEval**(*submod=None*)
Raw Code evaluation. The interpreter is chosen by the `modifier2`.

..note:: It does not return a valid uwsgi response, but a raw string (that may be an HTTP response).

class uwsgiconf.options.routing_modifiers.**ModifierXslt**(*submod=None*)
Invoke the The XSLT plugin.

class uwsgiconf.options.routing_modifiers.**ModifierV8**(*submod=None*)
Invoke the uWSGI V8 support.

class uwsgiconf.options.routing_modifiers.**ModifierGridfs**(*submod=None*)
Invoke the The GridFS plugin.

class uwsgiconf.options.routing_modifiers.**ModifierFastfunc**(*submod=None*)
Call the FastFuncs specified by the `modifier2` field.

class uwsgiconf.options.routing_modifiers.**ModifierGlusterfs**(*submod=None*)
Invoke the The GlusterFS plugin.

class uwsgiconf.options.routing_modifiers.**ModifierRados**(*submod=None*)
Invoke the The RADOS plugin.

class uwsgiconf.options.routing_modifiers.**ModifierManagePathInfo**(*submod=None*)
Standard WSGI request followed by the HTTP request body.

The `PATH_INFO` is automatically modified, removing the `SCRIPT_NAME` from it.

class uwsgiconf.options.routing_modifiers.**ModifierMessage**(*submod=None*)
Generic message passing (reserved).

class uwsgiconf.options.routing_modifiers.**ModifierMessageArray**(*submod=None*)
Array of char passing (reserved).

class uwsgiconf.options.routing_modifiers.**ModifierMessageMarshal**(*submod=None*)
Marshalled/serialzed object passing (reserved).

class uwsgiconf.options.routing_modifiers.**ModifierSnmp**(*submod=None*)
Identify a SNMP request/response (mainly via UDP).

class uwsgiconf.options.routing_modifiers.**ModifierRaw**(*submod=None*)
Corresponds to the `HTTP` string and signals that this is a raw HTTP response.

class uwsgiconf.options.routing_modifiers.**ModifierMulticastAnnounce**(*submod=None*)
Announce message.

class uwsgiconf.options.routing_modifiers.**ModifierMulticast**(*submod=None*)
Array of chars; a custom multicast message managed by uwsgi.

class uwsgiconf.options.routing_modifiers.**ModifierClusterNode**(*submod=None*)
Add/remove/enable/disable node from a cluster.

Add action requires a dict of at least 3 keys:
- hostname
- address
- workers

class uwsgiconf.options.routing_modifiers.**ModifierRemoteLogging**(*submod=None*)
Remote logging (clustering/multicast/unicast).

class uwsgiconf.options.routing_modifiers.**ModifierReload**(*submod=None*)
Graceful reload request.

class uwsgiconf.options.routing_modifiers.**ModifierReloadBrutal**(*submod=None*)
Brutal reload request.

class uwsgiconf.options.routing_modifiers.**ModifierConfigFromNode**(*submod=None*)
Request configuration data from a uwsgi node (even via multicast).

class uwsgiconf.options.routing_modifiers.**ModifierPing**(*submod=None*)
PING-PONG. Useful for cluster health check.

SUB_PING = 0
Request.

SUB_PONG = 1
Response.

class uwsgiconf.options.routing_modifiers.**ModifierEcho**(*submod=None*)
ECHO service.

class uwsgiconf.options.routing_modifiers.**ModifierLegionMsg**(*submod=None*)
Legion msg (UDP, the body is encrypted).

class uwsgiconf.options.routing_modifiers.**ModifierSignal**(*submod=None*)
uwsgi_signal framework (payload is optional).

Note: `modifier2` is the signal num.

---

class uwsgiconf.options.routing_modifiers.**ModifierCache**(*submod=None*)
Cache operations.

SUB_GET = 0
Simple cache get for values not bigger than 64k.

SUB_SET = 1
Simple cache set for values not bigger than 64k.

SUB_DELETE = 2
Simple cache del.

SUB_DICT_BASED = 3
Simple dict based get command.

SUB_STREAM = 5
Get and stream.

SUB_DUMP = 6
Dump the whole cache.

SUB_MAGIC = 17
Magic interface for plugins remote access.

class uwsgiconf.options.routing_modifiers.**ModifierCorerouterSignal**(*submod=None*)
Special modifier for signaling corerouters about special conditions.

class uwsgiconf.options.routing_modifiers.**ModifierRpc**(*submod=None*)
RPC. The packet is an uwsgi array where
- the first item - the name of the function
- the following - the args

SUB_DEFAULT = 0
Return uwsgi header + rpc response.

SUB_RAW = 1
Return raw rpc response, uwsgi header included, if available.

SUB_USE_PATH_INFO = 2
Split PATH_INFO to get func name and args and return as HTTP response with content_type as application/binary or Accept request header (if different from *).

SUB_XMLRPC = 3
Set xmlrpc wrapper (requires libxml2).

SUB_JSONRPC = 4
Set jsonrpc wrapper (requires libjansson).

SUB_DICT = 5
Used in uwsgi response to signal the response is a uwsgi dictionary followed by the body (the dictionary must contains a CONTENT_LENGTH key).

class uwsgiconf.options.routing_modifiers.**ModifierPersistentClose**(*submod=None*)
Close mark for persistent connections.

class uwsgiconf.options.routing_modifiers.**ModifierSubscription**(*submod=None*)
Subscription packet. See subscriptions.

class uwsgiconf.options.routing_modifiers.**ModifierExample**(*submod=None*)
Modifier used in dummy example plugin.

class uwsgiconf.options.routing_modifiers.**ModifierResponse**(*submod=None*)
Generic response. Request dependent.

Example: a spooler response set 0 for a failed spool or 1 for a successful one.

Subjects

class uwsgiconf.options.routing_subjects.**SubjectCustom**(*subject*, *, *negate=False*)

Represents a routing subject that supports various check.

Parameters

- **subject** (*Var|str*) – Handwritten subject or a Var heir representing it.
- **negate** (*bool*) – Use to negate subject for rule. .. note:: You can also use tilde (~) instead of this argument for nagation.

exists()

Check if the subject exists in the filesystem.

isfile()

Check if the subject is a file.

isdir()

Check if the subject is a directory.

islink()

Check if the subject is a link.

isexec()

Check if the subject is an executable file.

islord()

Check if the subject is a Legion Lord.

contains_ipv4()

Check if the subject is ip v4.

contains_ipv6()

Check if the subject is ip v6.

eq(*val*)

Check if the subject is equal to the specified pattern.

ge(*val*)

Check if the subject is greater than or equal to the specified pattern.

le(*val*)

Check if the subject is less than or equal to the specified pattern.

gt(*val*)

Check if the subject is greater than the specified pattern.

lt(*val*)

Check if the subject is less than the specified pattern.

startswith(*val*)

Check if the subject starts with the specified pattern.

endswith(*val*)

Check if the subject ends with the specified pattern.

matches(*regexp*)

Check if the subject matches the specified regexp.

isempty()

Check if the subject is empty.

contains(*val*)

Check if the subject contains the specified pattern.

class uwsgiconf.options.routing_subjects.**SubjectPathInfo**(*regexp*)
Default subject, maps to PATH_INFO.

class uwsgiconf.options.routing_subjects.**SubjectRequestUri**(*regexp*)
Checks REQUEST_URI for a value.

class uwsgiconf.options.routing_subjects.**SubjectQueryString**(*regexp*)
Checks QUERY_STRING for a value.

class uwsgiconf.options.routing_subjects.**SubjectRemoteAddr**(*regexp*)
Checks REMOTE_ADDR for a value.

class uwsgiconf.options.routing_subjects.**SubjectRemoteUser**(*regexp*)
Checks REMOTE_USER for a value.

class uwsgiconf.options.routing_subjects.**SubjectHttpHost**(*regexp*)
Checks HTTP_HOST for a value.

class uwsgiconf.options.routing_subjects.**SubjectHttpReferer**(*regexp*)
Checks HTTP_REFERER for a value.

class uwsgiconf.options.routing_subjects.**SubjectHttpUserAgent**(*regexp*)
Checks HTTP_USER_AGENT for a value.

class uwsgiconf.options.routing_subjects.**SubjectStatus**(*regexp*)
Checks HTTP response status code.

> **Warning:** Not available in the request chain.

Variables and Functions

class uwsgiconf.options.routing_vars.**Var**(*name: str*)

class uwsgiconf.options.routing_vars.**Func**(*name: str*)

class uwsgiconf.options.routing_vars.**VarGeoip**(*name: str*)
Returns Geoip data.

http://uwsgi.readthedocs.io/en/latest/GeoIP.html

vars_country = ['country_code', 'country_code3', 'country_name']
Keys avaiable for country database.

vars_city = ['continent', 'country_code', 'country_code3', 'country_name', 'region', '
Keys avaiable for city database.

class uwsgiconf.options.routing_vars.**VarRequest**(*name: str*)
Returns request variable. Examples: PATH_INFO, SCRIPT_NAME, REQUEST_METHOD.

class uwsgiconf.options.routing_vars.**VarMetric**(*name: str*)
Returns metric (see monitoring) variable.

class uwsgiconf.options.routing_vars.**VarCookie**(*name: str*)
Returns cookie variable

class uwsgiconf.options.routing_vars.**VarQuery**(*name: str*)
Returns query string variable.

class uwsgiconf.options.routing_vars.**VarUwsgi**(*name: str*)
Returns internal uWSGI information.
Supported variables:

- wid
- pid
- uuid
- status

class uwsgiconf.options.routing_vars.**VarTime**(*name: str*)
Returns time/date in various forms.
Supported variables:

- unix

class uwsgiconf.options.routing_vars.**VarHttptime**(*name: str*)
Returns http date adding the numeric argument (if specified) to the current time (use empty arg for current server time).

class uwsgiconf.options.routing_vars.**FuncMime**(*name: str*)
Returns mime type of a variable.

class uwsgiconf.options.routing_vars.**FuncMath**(*name: str*)
Perform a math operation. Example: CONTENT_LENGTH+1

Supported operations: + - * /

> **Warning:** Requires matheval support.

class uwsgiconf.options.routing_vars.**FuncBase64**(*name: str*)
Encodes the specified var in base64

class uwsgiconf.options.routing_vars.**FuncHex**(*name: str*)
Encodes the specified var in hex.

class uwsgiconf.options.routing_vars.**FuncUpper**(*name: str*)
Uppercase the specified var.

class uwsgiconf.options.routing_vars.**FuncLower**(*name: str*)
Lowercase the specified var.

class uwsgiconf.options.routing.**RouteRule**(*action*, *subject=None*, *stage=''*)
Represents a routing rule.

Parameters

- **action** (RouteAction) – Action (or transformation) to perfrom. See .actions and .transforms.
- **subject** (*SubjectCustom|SubjectBuiltin|str*) – Subject to verify before action is performed. See .subjects.
  - String values are automatically transformed into subjects.path_info.
  - If None action is performed always w/o subject check.
- **stage** (*str*) – Stage on which the action needs to be performed. See .stages.

class vars
Routing variables.

cookie
alias of *uwsgiconf.options.routing_vars.VarCookie*

geoip
alias of *uwsgiconf.options.routing_vars.VarGeoip*

httptime
alias of *uwsgiconf.options.routing_vars.VarHttptime*

metric
alias of *uwsgiconf.options.routing_vars.VarMetric*

query
alias of *uwsgiconf.options.routing_vars.VarQuery*

request
alias of *uwsgiconf.options.routing_vars.VarRequest*

time
alias of *uwsgiconf.options.routing_vars.VarTime*

uwsgi
alias of *uwsgiconf.options.routing_vars.VarUwsgi*

class var_functions
Functions that can be applied to variables.

base64
alias of *uwsgiconf.options.routing_vars.FuncBase64*

hex
alias of *uwsgiconf.options.routing_vars.FuncHex*

lower
alias of *uwsgiconf.options.routing_vars.FuncLower*

math
alias of *uwsgiconf.options.routing_vars.FuncMath*

mime
alias of *uwsgiconf.options.routing_vars.FuncMime*

upper
alias of *uwsgiconf.options.routing_vars.FuncUpper*

class stages
During the request cycle, various stages (aka chains) are processed.

Chains can be "recursive". A recursive chain can be called multiple times in a request cycle.

REQUEST = ''
Applied before the request is passed to the plugin.

ERROR = 'error'
Applied as soon as an HTTP status code is generate. **Recursive chain**.

RESPONSE = 'response'
Applied after the last response header has been generated (just before sending the body).

FINAL = 'final'
Applied after the response has been sent to the client.

class subjects
Routing subjects. These can be request's variables or other entities.

Note: Non-custom subjects can be pre-optimized (during startup) and should be used for performance reasons.

---

custom
alias of *uwsgiconf.options.routing_subjects.SubjectCustom*

http_host
alias of *uwsgiconf.options.routing_subjects.SubjectHttpHost*

http_referer
alias of *uwsgiconf.options.routing_subjects.SubjectHttpReferer*

http_user_agent
alias of *uwsgiconf.options.routing_subjects.SubjectHttpUserAgent*

path_info
alias of *uwsgiconf.options.routing_subjects.SubjectPathInfo*

query_string
alias of *uwsgiconf.options.routing_subjects.SubjectQueryString*

remote_addr
alias of *uwsgiconf.options.routing_subjects.SubjectRemoteAddr*

remote_user
alias of *uwsgiconf.options.routing_subjects.SubjectRemoteUser*

request_uri
alias of *uwsgiconf.options.routing_subjects.SubjectRequestUri*

status
alias of *uwsgiconf.options.routing_subjects.SubjectStatus*

class transforms
A transformation is like a filter applied to the response generated by your application.

Transformations can be chained (the output of a transformation will be the input of the following one) and can completely overwrite response headers.

- http://uwsgi.readthedocs.io/en/latest/Transformations.html

chunked
alias of *uwsgiconf.options.routing_actions.ActionChunked*

fix_content_len
alias of *uwsgiconf.options.routing_actions.ActionFixContentLen*

flush
alias of *uwsgiconf.options.routing_actions.ActionFlush*

gzip
alias of *uwsgiconf.options.routing_actions.ActionGzip*

template
alias of *uwsgiconf.options.routing_actions.ActionTemplate*

to_file
alias of *uwsgiconf.options.routing_actions.ActionToFile*

upper
alias of *uwsgiconf.options.routing_actions.ActionUpper*

class actions
Actions available for routing rules.

Values returned by actions:

- `NEXT` - continue to the next rule

- `CONTINUE` - stop scanning the internal routing table and run the request
- `BREAK` - stop scanning the internal routing table and close the request
- `GOTO x` - go to rule `x`

add_var_cgi
alias of *uwsgiconf.options.routing_actions.ActionAddVarCgi*

add_var_log
alias of *uwsgiconf.options.routing_actions.ActionAddVarLog*

alarm
alias of *uwsgiconf.options.routing_actions.ActionAlarm*

auth_basic
alias of *uwsgiconf.options.routing_actions.ActionAuthBasic*

auth_ldap
alias of *uwsgiconf.options.routing_actions.AuthLdap*

dir_change
alias of *uwsgiconf.options.routing_actions.ActionDirChange*

do_break
alias of *uwsgiconf.options.routing_actions.ActionDoBreak*

do_continue
alias of *uwsgiconf.options.routing_actions.ActionDoContinue*

do_goto
alias of *uwsgiconf.options.routing_actions.ActionDoGoto*

fix_var_path_info
alias of *uwsgiconf.options.routing_actions.ActionFixVarPathInfo*

header_add
alias of *uwsgiconf.options.routing_actions.ActionHeaderAdd*

header_remove
alias of *uwsgiconf.options.routing_actions.ActionHeaderRemove*

headers_off
alias of *uwsgiconf.options.routing_actions.ActionHeadersOff*

headers_reset
alias of *uwsgiconf.options.routing_actions.ActionHeadersReset*

log
alias of *uwsgiconf.options.routing_actions.ActionLog*

offload_off
alias of *uwsgiconf.options.routing_actions.ActionOffloadOff*

redirect
alias of *uwsgiconf.options.routing_actions.ActionRedirect*

rewrite
alias of *uwsgiconf.options.routing_actions.ActionRewrite*

route_external
alias of *uwsgiconf.options.routing_actions.ActionRouteExternal*

route_uwsgi
alias of *uwsgiconf.options.routing_actions.ActionRouteUwsgi*

send
alias of *uwsgiconf.options.routing_actions.ActionSend*

serve_static
alias of *uwsgiconf.options.routing_actions.ActionServeStatic*

set_harakiri
alias of *uwsgiconf.options.routing_actions.ActionSetHarakiri*

set_script_file
alias of *uwsgiconf.options.routing_actions.ActionSetScriptFile*

set_uwsgi_process_name
alias of *uwsgiconf.options.routing_actions.ActionSetUwsgiProcessName*

set_var_document_root
alias of *uwsgiconf.options.routing_actions.ActionSetVarDocumentRoot*

set_var_path_info
alias of *uwsgiconf.options.routing_actions.ActionSetVarPathInfo*

set_var_remote_addr
alias of *uwsgiconf.options.routing_actions.ActionSetVarRemoteAddr*

set_var_remote_user
alias of *uwsgiconf.options.routing_actions.ActionSetVarRemoteUser*

set_var_request_method
alias of *uwsgiconf.options.routing_actions.ActionSetVarRequestMethod*

set_var_request_uri
alias of *uwsgiconf.options.routing_actions.ActionSetVarRequestUri*

set_var_script_name
alias of *uwsgiconf.options.routing_actions.ActionSetVarScriptName*

set_var_uwsgi_appid
alias of *uwsgiconf.options.routing_actions.ActionSetVarUwsgiAppid*

set_var_uwsgi_home
alias of *uwsgiconf.options.routing_actions.ActionSetVarUwsgiHome*

set_var_uwsgi_scheme
alias of *uwsgiconf.options.routing_actions.ActionSetVarUwsgiScheme*

signal
alias of *uwsgiconf.options.routing_actions.ActionSignal*

class uwsgiconf.options.routing.**Routing**(**args*, ***kwargs*)
Routing subsystem.

You can use the internal routing subsystem to dynamically alter the way requests are handled.

Note: Since 1.9

- http://uwsgi.readthedocs.io/en/latest/InternalRouting.html
- http://uwsgi.readthedocs.io/en/latest/Transformations.html

route_rule
alias of *RouteRule*

class routers
Dedicated routers, which can be used with *register_router()*.

http
alias of *uwsgiconf.options.routing_routers.RouterHttp*

https
alias of *uwsgiconf.options.routing_routers.RouterHttps*

ssl
alias of *uwsgiconf.options.routing_routers.RouterSsl*

fast
alias of *uwsgiconf.options.routing_routers.RouterFast*

raw
alias of *uwsgiconf.options.routing_routers.RouterRaw*

forkpty
alias of *uwsgiconf.options.routing_routers.RouterForkPty*

tuntap
alias of *uwsgiconf.options.routing_routers.RouterTunTap*

class modifiers
Routing modifiers.

- http://uwsgi.readthedocs.io/en/latest/Protocol.html

cache
alias of *uwsgiconf.options.routing_modifiers.ModifierCache*

cgi
alias of *uwsgiconf.options.routing_modifiers.ModifierCgi*

cluster_node
alias of *uwsgiconf.options.routing_modifiers.ModifierClusterNode*

config_from_node
alias of *uwsgiconf.options.routing_modifiers.ModifierConfigFromNode*

corerouter_signal
alias of *uwsgiconf.options.routing_modifiers.ModifierCorerouterSignal*

echo
alias of *uwsgiconf.options.routing_modifiers.ModifierEcho*

eval
alias of *uwsgiconf.options.routing_modifiers.ModifierEval*

example
alias of *uwsgiconf.options.routing_modifiers.ModifierExample*

fastfunc
alias of *uwsgiconf.options.routing_modifiers.ModifierFastfunc*

glusterfs
alias of *uwsgiconf.options.routing_modifiers.ModifierGlusterfs*

gridfs
alias of *uwsgiconf.options.routing_modifiers.ModifierGridfs*

jvm
alias of *uwsgiconf.options.routing_modifiers.ModifierJvm*

legion_msg
alias of *uwsgiconf.options.routing_modifiers.ModifierLegionMsg*

lua
alias of *uwsgiconf.options.routing_modifiers.ModifierLua*

manage
alias of *uwsgiconf.options.routing_modifiers.ModifierManage*

manage_path_info
alias of *uwsgiconf.options.routing_modifiers.ModifierManagePathInfo*

message
alias of *uwsgiconf.options.routing_modifiers.ModifierMessage*

message_array
alias of *uwsgiconf.options.routing_modifiers.ModifierMessageArray*

message_marshal
alias of *uwsgiconf.options.routing_modifiers.ModifierMessageMarshal*

mono
alias of *uwsgiconf.options.routing_modifiers.ModifierMono*

multicast
alias of *uwsgiconf.options.routing_modifiers.ModifierMulticast*

multicast_announce
alias of *uwsgiconf.options.routing_modifiers.ModifierMulticastAnnounce*

persistent_close
alias of *uwsgiconf.options.routing_modifiers.ModifierPersistentClose*

php
alias of *uwsgiconf.options.routing_modifiers.ModifierPhp*

ping
alias of *uwsgiconf.options.routing_modifiers.ModifierPing*

psgi
alias of *uwsgiconf.options.routing_modifiers.ModifierPsgi*

rack
alias of *uwsgiconf.options.routing_modifiers.ModifierRack*

rados
alias of *uwsgiconf.options.routing_modifiers.ModifierRados*

raw
alias of *uwsgiconf.options.routing_modifiers.ModifierRaw*

reload
alias of *uwsgiconf.options.routing_modifiers.ModifierReload*

reload_brutal
alias of *uwsgiconf.options.routing_modifiers.ModifierReloadBrutal*

remote_logging
alias of *uwsgiconf.options.routing_modifiers.ModifierRemoteLogging*

response
alias of *uwsgiconf.options.routing_modifiers.ModifierResponse*

rpc
alias of *uwsgiconf.options.routing_modifiers.ModifierRpc*

signal
alias of *uwsgiconf.options.routing_modifiers.ModifierSignal*

snmp
alias of *uwsgiconf.options.routing_modifiers.ModifierSnmp*

spooler
alias of *uwsgiconf.options.routing_modifiers.ModifierSpooler*

subscription
alias of *uwsgiconf.options.routing_modifiers.ModifierSubscription*

symcall
alias of *uwsgiconf.options.routing_modifiers.ModifierSymcall*

v8
alias of *uwsgiconf.options.routing_modifiers.ModifierV8*

wsgi
alias of *uwsgiconf.options.routing_modifiers.ModifierWsgi*

xslt
alias of *uwsgiconf.options.routing_modifiers.ModifierXslt*

use_router(*router*, *, *force=None*)

Parameters

- **router** (RouterBase) – Dedicated router object. See *.routers*.
- **force** (*bool*) – All of the gateways (routers) has to be run under the master process, supplying this you can try to bypass this limit.

register_route(*route_rules*, *, *label=None*)
Registers a routing rule.

Parameters

- **route_rules** (*RouteRule|list[RouteRule]*) –
- **label** (*str*) – Label to mark the given set of rules. This can be used in conjunction with `do_goto` rule action.
  - http://uwsgi.readthedocs.io/en/latest/InternalRouting.html#goto

print_routing_rules()
Print out supported routing rules (actions, transforms, etc.).

set_error_page(*status: int*, *html_fpath: str*)
Add an error page (html) for managed 403, 404, 500 response.

Parameters

- **status** – HTTP status code.
- **html_fpath** – HTML page file path.

set_error_pages(*codes_map: dict = None*, *, *common_prefix: str = None*)
Add an error pages for managed 403, 404, 500 responses.

Shortcut for `.set_error_page()`.

Parameters

- **codes_map** – Status code mapped into an html filepath or just a filename if common_prefix is used.

  If not set, filename containing status code is presumed: 400.html, 500.html, etc.

- **common_prefix** – Common path (prefix) for all files.

set_geoip_params(*, *db_country=None*, *db_city=None*)
Sets GeoIP parameters.

- http://uwsgi.readthedocs.io/en/latest/GeoIP.html

Parameters

- **db_country** (*str*) – Country database file path.
- **db_city** (*str*) – City database file path. Example: `GeoLiteCity.dat`.

header_add(*name*, *value*)
Automatically add HTTP headers to response.

Parameters

- **name** (*str*) –
- **value** (*str*) –

header_remove(*value*)
Automatically remove specified HTTP header from the response.

Parameters **value** (*str*) –

header_collect(*name*, *target_var*, *, *pull=False*)
Store the specified response header in a request var (optionally removing it from the response).

Parameters

- **name** (*str*) –
- **target_var** (*str*) –
- **pull** (*bool*) – Whether to remove header from response.

3.8.14 Spooler

class uwsgiconf.options.spooler.**Spooler**(**args*, ***kwargs*)
Spooler.

Note: Supported on: Perl, Python, Ruby.

Note: Be sure the `spooler` plugin is loaded in your instance, but generally it is built in by default.

The Spooler is a queue manager built into uWSGI that works like a printing/mail system. You can enqueue massive sending of emails, image processing, video encoding, etc. and let the spooler do the hard work in background while your users get their requests served by normal workers.

http://uwsgi-docs.readthedocs.io/en/latest/Spooler.html

set_basic_params(*, *touch_reload: Union[str, List[str]] = None*, *quiet: bool = None*, *process_count: int = None*, *max_tasks: int = None*, *order_tasks: int = None*, *harakiri: int = None*, *change_dir: str = None*, *poll_interval: int = None*, *signal_as_task: bool = None*, *cheap: bool = None*, *base_dir: str = None*)

Parameters

- **touch_reload** – reload spoolers if the specified file is modified/touched
- **quiet** – Do not log spooler related messages.
- **process_count** – Set the number of processes for spoolers.
- **max_tasks** – Set the maximum number of tasks to run before recycling a spooler (to help alleviate memory leaks).
- **order_tasks** – Try to order the execution of spooler tasks (uses scandir instead of readdir).
- **harakiri** – Set harakiri timeout for spooler tasks.
- **change_dir** – chdir() to specified directory before each spooler task.
- **poll_interval** – Spooler poll frequency in seconds. Default: 30.
- **signal_as_task** – Treat signal events as tasks in spooler. To be used with `spooler-max-tasks`. If enabled spooler will treat signal events as task. Run signal handler will also increase the spooler task count.
- **cheap** – Use spooler cheap mode.
- **base_dir** – Base directory to prepend to *work_dir* argument of *.add()*.

add(*work_dir: Union[str, List[str]], *, external: bool = False*)

Run a spooler on the specified directory.

Parameters

- **work_dir** – Spooler working directory path or it's name if *base_dir* argument of *spooler.set_basic_params()* is set.

  Note: Placeholders can be used to build paths, e.g.: {project_runtime_dir}/spool/ See `Section.project_name` and `Section.runtime_dir`.

- **external** – map spoolers requests to a spooler directory managed by an external instance

3.8.15 Statics

class uwsgiconf.options.statics.**Statics**(**args*, ***kwargs*)

Statics.

Unfortunately you cannot live without serving static files via some protocol (HTTP, SPDY or something else). Fortunately uWSGI has a wide series of options and micro-optimizations for serving static files.

Note: This subsystem automatically honours the `If-Modified-Since` HTTP request header.

- http://uwsgi.readthedocs.io/en/latest/StaticFiles.html

DIR_DOCUMENT_ROOT = 'docroot'
Used to check for static files in the requested DOCUMENT_ROOT. Pass into `static_dir`.

class expiration_criteria
Expiration criteria (subjects) to use with `.add_expiration_rule()`.

FILENAME = 'filename'
Allows setting the Expires header for the specified file name pattern.

MIME_TYPE = 'type'
Allows setting the Expires header for the specified MIME type.

PATH_INFO = 'path-info'
Allows setting the Expires header for the specified `PATH_INFO` pattern.

REQUEST_URI = 'uri'
Allows setting the Expires header for the specified `REQUEST_URI` pattern.

class transfer_modes
File transfer (serving) modes.

With this, uWSGI will only generate response headers and the web server will be delegated to transferring the physical file.

- http://uwsgi.readthedocs.io/en/latest/StaticFiles.html#transfer-modes

SENDFILE = 'x-sendfile'
Use `X-Sendfile` mode. Apache.

ACCEL_REDIRECT = 'x-accel-redirect'
Use `X-Accel-Redirect` mode. Nginx.

set_basic_params(*, *static_dir=None*, *index_file=None*, *mime_file=None*, *skip_ext=None*, *transfer_mode=None*)

Parameters

- **static_dir** (*str|list[str]*) – Check for static files in the specified directory.

  Note: Use `DIR_DOCUMENT_ROOT` constant to serve files under `DOCUMENT_ROOT`.

- **index_file** (*str|list[str]*) – Search for specified file if a directory is requested.

  Example: `index.html`

- **mime_file** (*str|list[str]*) – Set mime types file path to extend uWSGI builtin list.

  Default: `/etc/mime.types` or `/etc/apache2/mime.types`.

- **skip_ext** (*str|list[str]*) – Skip specified extension from static file checks.

  Example: add `.php` to not serve it as static.

- **transfer_mode** (*str*) – Set static file serving (transfer) mode.

  See `.transfer_modes`.

> **Note:** Another option is to specify `count_offload` in `.workers.set_thread_params()`.

register_static_map(*mountpoint*, *target*, *, *retain_resource_path=False*, *safe_target=False*)
Allows mapping mountpoint to a static directory (or file).

- http://uwsgi.readthedocs.io/en/latest/StaticFiles.html#mode-3-using-static-file-mount-points

Parameters

- **mountpoint** (*str*) –
- **target** (*str*) –
- **retain_resource_path** (*bool*) – Append the requested resource to the docroot.

  Example: if `/images` maps to `/var/www/img` requested `/images/logo.png` will be served from:

  - `True`: `/var/www/img/images/logo.png`
  - `False`: `/var/www/img/logo.png`
- **safe_target** (*bool*) – Skip security checks if the file is under the specified path.

  Whether to consider resolved (real) target a safe one to serve from.

  - http://uwsgi.readthedocs.io/en/latest/StaticFiles.html#security

add_expiration_rule(*criterion*, *value*, *, *timeout*, *use_mod_time=False*)
Adds statics expiration rule based on a criterion.

Parameters

- **criterion** (*str*) – Criterion (subject) to base expiration on.

  See `.expiration_criteria`.
- **value** (*str|list[str]*) – Value to test criteria upon.

  > **Note:** Usually a regular expression.
- **timeout** (*int*) – Number of seconds to expire after.
- **use_mod_time** (*bool*) – Base on file modification time instead of the current time.

set_paths_caching_params(*, *timeout=None*, *cache_name=None*)
Use the uWSGI caching subsystem to store mappings from URI to filesystem paths.

- http://uwsgi.readthedocs.io/en/latest/StaticFiles.html#caching-paths-mappings-resolutions

Parameters

- **timeout** (*int*) – Amount of seconds to put resolved paths in the uWSGI cache.
- **cache_name** (*str*) – Cache name to use for static paths.

3.8.16 Subscriptions

Balancing Algorithms

class uwsgiconf.options.subscriptions_algos.**BalancingAlgorithm**(**args*)

class uwsgiconf.options.subscriptions_algos.**BalancingAlgorithmWithBackup**(*backup_level=None*)

class uwsgiconf.options.subscriptions_algos.**WeightedRoundRobin**(*backup_level=None*)
Weighted round robin algorithm with backup support. The default algorithm.

class uwsgiconf.options.subscriptions_algos.**LeastReferenceCount**(*backup_level=None*)
Least reference count algorithm with backup support.

class uwsgiconf.options.subscriptions_algos.**WeightedLeastReferenceCount**(*backup_level=None*)
Weighted least reference count algorithm with backup support.

class uwsgiconf.options.subscriptions_algos.**IpHash**(*backup_level=None*)
IP hash algorithm with backup support.

class uwsgiconf.options.subscriptions.**Subscriptions**(**args*, ***kwargs*)
This allows some uWSGI instances to announce their presence to subscriptions managing server, which in its turn can address those nodes (e.g. delegate request processing to them) and automatically remove dead nodes from the pool.

Some routers provide subscription server functionality. See *.routing.routers*.

Note: Subscription system in many ways relies on Master Process.

Warning: The subscription system is meant for "trusted" networks. All of the nodes in your network can potentially make a total mess with it.

- http://uwsgi.readthedocs.io/en/latest/SubscriptionServer.html

class algorithms
Balancing algorithms available to use with `subscribe`.

ip_hash
alias of *uwsgiconf.options.subscriptions_algos.IpHash*

least_reference_count
alias of *uwsgiconf.options.subscriptions_algos.LeastReferenceCount*

weighted_least_reference_count
alias of *uwsgiconf.options.subscriptions_algos.WeightedLeastReferenceCount*

weighted_round_robin
alias of *uwsgiconf.options.subscriptions_algos.WeightedRoundRobin*

set_server_params(***, *client_notify_address=None*, *mountpoints_depth=None*, *require_vassal=None*, *tolerance=None*, *tolerance_inactive=None*, *key_dot_split=None*)
Sets subscription server related params.

Parameters

- **client_notify_address** (*str*) – Set the notification socket for subscriptions. When you subscribe to a server, you can ask it to "acknowledge" the acceptance of your request. pointing address (Unix socket or UDP), on which your instance will bind and the subscription server will send acknowledgements to.
- **mountpoints_depth** (*int*) – Enable support of mountpoints of certain depth for subscription system.
  - http://uwsgi-docs.readthedocs.io/en/latest/SubscriptionServer.html#mountpoints-uwsgi-2-1
- **require_vassal** (*bool*) – Require a vassal field (see `subscribe`) from each subscription.
- **tolerance** (*int*) – Subscription reclaim tolerance (seconds).
- **tolerance_inactive** (*int*) – Subscription inactivity tolerance (seconds).
- **key_dot_split** (*bool*) – Try to fallback to the next part in (dot based) subscription key. Used, for example, in SNI.

set_server_verification_params(*, *digest_algo=None*, *dir_cert=None*, *tolerance=None*, *no_check_uid=None*, *dir_credentials=None*, *pass_unix_credentials=None*)

Sets peer verification params for subscription server.

These are for secured subscriptions.

Parameters

- **digest_algo** (*str*) – Digest algorithm. Example: SHA1

  Note: Also requires `dir_cert` to be set.

- **dir_cert** (*str*) – Certificate directory.

  Note: Also requires `digest_algo` to be set.

- **tolerance** (*int*) – Maximum tolerance (in seconds) of clock skew for secured subscription system. Default: 24h.
- **no_check_uid** (*str|int|list[str|int]*) – Skip signature check for the specified uids when using unix sockets credentials.
- **dir_credentials** (*str|list[str]*) – Directories to search for subscriptions key credentials.
- **pass_unix_credentials** (*bool*) – Enable management of SCM_CREDENTIALS in subscriptions UNIX sockets.

set_client_params(*, *start_unsubscribed=None*, *clear_on_exit=None*, *unsubscribe_on_reload=None*, *announce_interval=None*)

Sets subscribers related params.

Parameters

- **start_unsubscribed** (*bool*) – Configure subscriptions but do not send them. .. note:: Useful with master FIFO.
- **clear_on_exit** (*bool*) – Force clear instead of unsubscribe during shutdown.

- **unsubscribe_on_reload** (*bool*) – Force unsubscribe request even during graceful reload.
- **announce_interval** (*int*) – Send subscription announce at the specified interval. Default: 10 master cycles.

subscribe(*server=None*, ***, *key=None*, *address=None*, *address_vassal=None*, *balancing_weight=None*, *balancing_algo=None*, *modifier=None*, *signing=None*, *check_file=None*, *protocol=None*, *sni_cert=None*, *sni_key=None*, *sni_client_ca=None*)

Registers a subscription intent.

Parameters

- **server** (*str*) – Subscription server address (UDP or UNIX socket).

  Examples:

  - 127.0.0.1:7171
- **key** (*str*) – Key to subscribe. Generally the domain name (+ optional '/< mountpoint>'). Examples:
  - mydomain.it/foo
  - mydomain.it/foo/bar (requires `mountpoints_depth=2`)
  - mydomain.it
  - ubuntu64.local:9090
- **address** (*str*) – Address to subscribe (the value for the key) or zero-based internal socket number (integer).
- **address** – Vassal node address.
- **balancing_weight** (*int*) – Load balancing value. Default: 1.
- **balancing_algo** – Load balancing algorithm to use. See `balancing_algorithms` .. note:: Since 2.1
- **modifier** (*Modifier*) – Routing modifier object. See `.routing.modifiers`
- **signing** (*list|tuple*) – Signing basics, expects two elements list/tuple: (signing_algorithm, key).

  Examples:

  - SHA1:idlessh001
- **check_file** (*str*) – If this file exists the subscription packet is sent, otherwise it is skipped.
- **protocol** (*str*) – the protocol to use, by default it is `uwsgi`. See `.networking.socket_types`.

  Note: Since 2.1

- **sni_cert** (*str*) – Certificate file to use for SNI proxy management. * http://uwsgi.readthedocs.io/en/latest/SNI.html#subscription-system-and-sni
- **sni_key** (*str*) – sni_key Key file to use for SNI proxy management. * http://uwsgi.readthedocs.io/en/latest/SNI.html#subscription-system-and-sni

- **sni_client_ca** (*str*) – Ca file to use for SNI proxy management. * http://uwsgi.readthedocs.io/en/latest/SNI.html#subscription-system-and-sni

3.8.17 Workers

Cheapening

class uwsgiconf.options.workers_cheapening.**Algo**(**args*, ***kwargs*)

class uwsgiconf.options.workers_cheapening.**AlgoSpare**(**args*, ***kwargs*)

The **default** algorithm.

If all workers are busy for a certain amount of time seconds then uWSGI will spawn new workers. When the load is gone it will begin stopping processes one at a time.

- http://uwsgi.readthedocs.io/en/latest/Cheaper.html#spare-cheaper-algorithm

set_basic_params(*, *check_interval_overload=None*)

Parameters **check_interval_overload** (*int*) – Interval (sec) to wait after all workers are busy before new worker spawn.

class uwsgiconf.options.workers_cheapening.**AlgoSpare2**(**args*, ***kwargs*)

This algorithm is similar to `spare`, but suitable for large scale by increase workers faster (before overload) and decrease them slower.

- http://uwsgi.readthedocs.io/en/latest/Cheaper.html#spare2-cheaper-algorithm

set_basic_params(*, *check_interval_idle=None*)

Parameters **check_interval_idle** (*int*) – Decrease workers after specified idle. Default: 10.

class uwsgiconf.options.workers_cheapening.**AlgoQueue**(**args*, ***kwargs*)

If the socket's listen queue has more than `cheaper_overload` requests waiting to be processed, uWSGI will spawn new workers.

If the backlog is lower it will begin killing processes one at a time.

- http://uwsgi.readthedocs.io/en/latest/Cheaper.html#backlog-cheaper-algorithm

set_basic_params(*, *check_num_overload=None*)

Parameters **check_num_overload** (*int*) – Number of backlog items in queue.

class uwsgiconf.options.workers_cheapening.**AlgoBusyness**(**args*, ***kwargs*)

Algorithm adds or removes workers based on average utilization for a given time period. It's goal is to keep more workers than the minimum needed available at any given time, so the app will always have capacity for new requests.

- http://uwsgi.readthedocs.io/en/latest/Cheaper.html#busyness-cheaper-algorithm

---

Note: Requires `cheaper_busyness` plugin.

---

set_basic_params(*, *check_interval_busy=None*, *busy_max=None*, *busy_min=None*, *idle_cycles_max=None*, *idle_cycles_penalty=None*, *verbose=None*)

Parameters

- **check_interval_busy** (*int*) – Interval (sec) to check worker busyness.
- **busy_max** (*int*) – Maximum busyness (percents). Every time the calculated busyness is higher than this value, uWSGI will spawn new workers. Default: 50.
- **busy_min** (*int*) – Minimum busyness (percents). If busyness is below this value, the app is considered in an "idle cycle" and uWSGI will start counting them.

Once we reach needed number of idle cycles uWSGI will kill one worker. Default: 25.

- **idle_cycles_max** (*int*) – This option tells uWSGI how many idle cycles are allowed before stopping a worker.
- **idle_cycles_penalty** (*int*) – Number of idle cycles to add to `idle_cycles_max` in case worker spawned too early. Default is 1.
- **verbose** (*bool*) – Enables debug logs for this algo.

set_emergency_params(*, *workers_step=None*, *idle_cycles_max=None*, *queue_size=None*, *queue_nonzero_delay=None*)

Sets busyness algorithm emergency workers related params.

Emergency workers could be spawned depending upon uWSGI backlog state.

Note: These options are Linux only.

Parameters

- **workers_step** (*int*) – Number of emergency workers to spawn. Default: 1.
- **idle_cycles_max** (*int*) – Idle cycles to reach before stopping an emergency worker. Default: 3.
- **queue_size** (*int*) – Listen queue (backlog) max size to spawn an emergency worker. Default: 33.
- **queue_nonzero_delay** (*int*) – If the request listen queue is > 0 for more than given amount of seconds new emergency workers will be spawned. Default: 60.

class uwsgiconf.options.workers_cheapening.**AlgoManual**(**args*, ***kwargs*)

Algorithm allows to adjust number of workers using Master FIFO commands.

- http://uwsgi.readthedocs.io/en/latest/MasterFIFO.html#available-commands

class uwsgiconf.options.workers_cheapening.**Cheapening**(**args*, ***kwargs*)

uWSGI provides the ability to dynamically scale the number of running workers (adaptive process spawning) via pluggable algorithms.

Note: This uses master process.

class algorithms

Algorithms available to use with `cheaper_algorithm`.

busyness

alias of *AlgoBusyness*

manual

alias of *AlgoManual*

queue

alias of *AlgoQueue*

spare

alias of *AlgoSpare*

spare2
alias of *AlgoSpare2*

set_basic_params(*, *spawn_on_request=None*, *cheaper_algo=None*, *workers_min=None*, *workers_startup=None*, *workers_step=None*)

Parameters

- **spawn_on_request** (*bool*) – Spawn workers only after the first request.
- **cheaper_algo** (*Algo*) – The algorithm object to be used used for adaptive process spawning. Default: `spare`. See `.algorithms`.
- **workers_min** (*int*) – Minimal workers count. Enables cheaper mode (adaptive process spawning).

  Note: Must be lower than max workers count.

- **workers_startup** (*int*) – The number of workers to be started when starting the application. After the app is started the algorithm can stop or start workers if needed.
- **workers_step** (*int*) – Number of additional processes to spawn at a time if they are needed,

set_memory_limits(*, *rss_soft=None*, *rss_hard=None*)
Sets worker memory limits for cheapening.

Parameters

- **rss_soft** (*int*) – Don't spawn new workers if total resident memory usage of all workers is higher than this limit in bytes.

  Warning: This option expects memory reporting enabled: `.logging.set_basic_params(memory_report=1)`

- **rss_hard** (*int*) – Try to stop workers if total workers resident memory usage is higher that thi limit in bytes.

print_alorithms()
Print out enabled cheaper algorithms.

class uwsgiconf.options.workers.**MuleFarm**(*name: str, mule_numbers: Union[int, List[int]]*)
Represents a mule farm.

Parameters

- **name** – Farm alias.
- **mule_numbers** – Total mules on farm count, or a list of mule numbers.

class uwsgiconf.options.workers.**Workers**(**args*, ***kwargs*)
Workers aka [working] processes.

mule_farm
alias of *MuleFarm*

set_basic_params(*, *count: int = None*, *touch_reload: Union[str, List[str]] = None*, *touch_chain_reload: Union[str, List[str]] = None*, *zombie_reaper: bool = None*, *limit_addr_space: int = None*, *limit_count: int = None*, *cpu_affinity: int = None*)

Parameters

- **count** – Spawn the specified number of workers (processes). Set the number of workers for preforking mode. This is the base for easy and safe concurrency in your app. More workers you add, more concurrent requests you can manage.

  Each worker corresponds to a system process, so it consumes memory, choose carefully the right number. You can easily drop your system to its knees by setting a too high value.

  Setting `workers` to a ridiculously high number will **not** magically make your application web scale - quite the contrary.

- **touch_reload** – Trigger reload of (and only) workers if the specified file is modified/touched.

- **touch_chain_reload** – Trigger chain workers reload on file touch. When in lazy/lazy_apps mode, you can simply destroy a worker to force it to reload the application code. A new reloading system named "chain reload", allows you to reload one worker at time (opposed to the standard way where all of the workers are destroyed in bulk)

  - http://uwsgi-docs.readthedocs.io/en/latest/articles/TheArtOfGracefulReloading.html#chain-reloading-lazy-apps

- **zombie_reaper** – Call waitpid(-1,...) after each request to get rid of zombies. Enables reaper mode. After each request the server will call `waitpid(-1)` to get rid of zombie processes. If you spawn subprocesses in your app and you happen to end up with zombie processes all over the place you can enable this option. (It really would be better if you could fix your application's process spawning usage though.)

- **limit_addr_space** – Limit process address space (vsz) (in megabytes) Limits the address space usage of each uWSGI (worker) process using POSIX/UNIX `setrlimit()`. For example, `limit-as 256` will disallow uWSGI processes to grow over 256MB of address space. Address space is the virtual memory a process has access to. It does *not* correspond to physical memory. Read and understand this page before enabling this option: http://en.wikipedia.org/wiki/Virtual_memory

- **limit_count** – Limit the number of spawnable processes.

- **cpu_affinity** – number of cores for each worker (Linux only) Set the number of cores (CPUs) to allocate to each worker process.

  - **4 workers, 4 CPUs, affinity is 1,** each worker is allocated one CPU.
  - **4 workers, 2 CPUs, affinity is 1,** workers get one CPU each (0; 1; 0; 1).
  - **4 workers, 4 CPUs, affinity is 2,** workers get two CPUs each in a round-robin fashion (0, 1; 2, 3; 0, 1; 2; 3).
  - **8 workers, 4 CPUs, affinity is 3,** workers get three CPUs each in a round-robin fashion (0, 1, 2; 3, 0, 1; 2, 3, 0; 1, 2, 3; 0, 1, 2; 3, 0, 1; 2, 3, 0; 1, 2, 3).

run_command_as_worker(*command: str*, *, *after_post_fork_hook: bool = False*)

Run the specified command as worker.

Parameters

- **command** –

- **after_post_fork_hook** – Whether to run it after *post_fork* hook.

set_count_auto(*count: int = None*)
Sets workers count.

By default sets it to detected number of available cores

Parameters **count** –

set_thread_params(*enable: bool = None*, ***, *count: int = None*, *count_offload: int = None*, *stack_size: int = None*, *no_wait: bool = None*)
Sets threads related params.

Parameters

- **enable** – Enable threads in the embedded languages. This will allow to spawn threads in your app.

  Warning: Threads will simply *not work* if this option is not enabled. There will likely be no error, just no execution of your thread code.

- **count** – Run each worker in prethreaded mode with the specified number of threads per worker.

  Warning: Do not use with `gevent`.

  Note: Enables threads automatically.

- **count_offload** – Set the number of threads (per-worker) to spawn for offloading. Default: 0.

  These threads run such tasks in a non-blocking/evented way allowing for a huge amount of concurrency. Various components of the uWSGI stack are offload-friendly.

  Note: Try to set it to the number of CPU cores to take advantage of SMP.

  – http://uwsgi-docs.readthedocs.io/en/latest/OffloadSubsystem.html

- **stack_size** – Set threads stacksize.
- **no_wait** – Do not wait for threads cancellation on quit/reload.

set_mules_params(*mules: Union[int, List[int]] = None*, ***, *touch_reload: Union[str, List[str]] = None*, *harakiri_timeout: int = None*, *farms: List[uwsgiconf.options.workers.MuleFarm] = None*, *reload_mercy: int = None*, *msg_buffer: int = None*, *msg_buffer_recv: int = None*)
Sets mules related params.

http://uwsgi.readthedocs.io/en/latest/Mules.html

Mules are worker processes living in the uWSGI stack but not reachable via socket connections, that can be used as a generic subsystem to offload tasks.

Parameters

- **mules** – Add the specified mules or number of mules.
- **touch_reload** – Reload mules if the specified file is modified/touched.
- **harakiri_timeout** – Set harakiri timeout for mule tasks.
- **farms** – Mule farms list.

  Examples:

  - cls_mule_farm('first', 2)
  - cls_mule_farm('first', [4, 5])
- **reload_mercy** – Set the maximum time (in seconds) a mule can take to reload/shutdown. Default: 60.
- **msg_buffer** – Set mule message buffer size (bytes) given for mule message queue.
- **msg_buffer** – Set mule message recv buffer size (bytes).

set_reload_params(*, *min_lifetime: int = None*, *max_lifetime: int = None*, *max_requests: int = None*, *max_requests_delta: int = None*, *max_addr_space: int = None*, *max_rss: int = None*, *max_uss: int = None*, *max_pss: int = None*, *max_addr_space_forced: int = None*, *max_rss_forced: int = None*, *watch_interval_forced: int = None*, *mercy: int = None*)

Sets workers reload parameters.

Parameters

- **min_lifetime** – A worker cannot be destroyed/reloaded unless it has been alive for N seconds (default 60). This is an anti-fork-bomb measure. Since 1.9
- **max_lifetime** – Reload workers after this many seconds. Disabled by default. Since 1.9
- **max_requests** – Reload workers after the specified amount of managed requests (avoid memory leaks). When a worker reaches this number of requests it will get recycled (killed and restarted). You can use this option to "dumb fight" memory leaks.

  Also take a look at the `reload-on-as` and `reload-on-rss` options as they are more useful for memory leaks.

  Warning: The default min-worker-lifetime 60 seconds takes priority over *max-requests*.

  Do not use with benchmarking as you'll get stalls such as *worker respawning too fast !!! i have to sleep a bit (2 seconds)...*
- **max_requests_delta** – Add (worker_id * delta) to the max_requests value of each worker.
- **max_addr_space** – Reload a worker if its address space usage is higher than the specified value in megabytes.
- **max_rss** – Reload a worker if its physical unshared memory (resident set size) is higher than the specified value (in megabytes).
- **max_uss** – Reload a worker if Unique Set Size is higher than the specified value in megabytes.

Note: Linux only.

- **max_pss** – Reload a worker if Proportional Set Size is higher than the specified value in megabytes.

Note: Linux only.

- **max_addr_space_forced** – Force the master to reload a worker if its address space is higher than specified megabytes (in megabytes).
- **max_rss_forced** – Force the master to reload a worker if its resident set size memory is higher than specified in megabytes.
- **watch_interval_forced** – The memory collector [per-worker] thread memeory watch interval (seconds) used for forced reloads. Default: 3.
- **mercy** – Set the maximum time (in seconds) a worker can take before reload/shutdown. Default: 60.

set_reload_on_exception_params(*, *do_reload: bool = None*, *etype: str = None*, *evalue: str = None*, *erepr: str = None*)

Sets workers reload on exceptions parameters.

Parameters

- **do_reload** – Reload a worker when an exception is raised.
- **etype** – Reload a worker when a specific exception type is raised.
- **evalue** – Reload a worker when a specific exception value is raised.
- **erepr** – Reload a worker when a specific exception type+value (language-specific) is raised.

set_harakiri_params(*, *timeout: int = None*, *verbose: bool = None*, *disable_for_arh: bool = None*)

Sets workers harakiri parameters.

Parameters

- **timeout** – Harakiri timeout in seconds. Every request that will take longer than the seconds specified in the harakiri timeout will be dropped and the corresponding worker is thereafter recycled.
- **verbose** – Harakiri verbose mode. When a request is killed by Harakiri you will get a message in the uWSGI log. Enabling this option will print additional info (for example, the current syscall will be reported on Linux platforms).
- **disable_for_arh** – Disallow Harakiri killings during after-request hook methods.

set_zerg_server_params(*socket: str*, *, *clients_socket_pool: Union[str, List[str]] = None*)

Zerg mode. Zerg server params.

When your site load is variable, it would be nice to be able to add workers dynamically. Enabling Zerg mode you can allow zerg clients to attach to your already running server and help it in the work.

- http://uwsgi-docs.readthedocs.io/en/latest/Zerg.html

Parameters

- **socket** – Unix socket to bind server to.

  Examples:

  - unix socket - `/var/run/mutalisk`
  - Linux abstract namespace - `@nydus`

- **clients_socket_pool** – This enables Zerg Pools.

  Note: Expects master process.

  Accepts sockets that will be mapped to Zerg socket.

  - http://uwsgi-docs.readthedocs.io/en/latest/Zerg.html#zerg-pools

set_zerg_client_params(*server_sockets: Union[str, List[str]], *, use_fallback_socket: bool = None*)

Zerg mode. Zergs params.

Parameters

- **server_sockets** – Attaches zerg to a zerg server.
- **use_fallback_socket** – Fallback to normal sockets if the zerg server is not available

3.8.18 Python

class uwsgiconf.options.python.**Python**(**args*, ***kwargs*)

Python plugin options.

Note: By default the plugin does not initialize the GIL. This means your app-generated threads will not run. If you need threads, remember to enable them with `enable_threads`.

set_basic_params(**, version: Union[str, int] = -1, python_home: str = None, enable_threads: bool = None, search_path: str = None, python_binary: str = None, trace-backer_path: str = None, plugin_dir: str = None, os_env_reload: bool = None, optimization_level: int = None*)

Parameters

- **version** – Python version plugin supports.

  Example:

  - 3 - version 3
  - <empty> - version 2
  - <default> - version deduced by uwsgiconf

- **python_home** – Set python executable directory - PYTHONHOME/virtualenv.
- **enable_threads** (*bool*) – Enable threads in the embedded languages. This will allow to spawn threads in your app.

Warning: Threads will simply *not work* if this option is not enabled. There will likely be no error, just no execution of your thread code.

- **search_path** – Add directory (or an .egg or a glob) to the Python search path.

  Note: This can be specified up to 64 times.

- **python_binary** – Set python program name.
- **tracebacker_path** – Enable the uWSGI Python tracebacker. http://uwsgi-docs.readthedocs.io/en/latest/Tracebacker.html
- **plugin_dir** – Directory to search for plugin.
- **os_env_reload** – Force `os.environ` reloading for every request. Used to allow setting of `UWSGI_SETENV` for Python applications.
- **optimization_level** – Python optimization level (see `-O` argument). .. warning:: This may be dangerous for some apps.

set_app_args(**args*)

Sets `sys.argv` for python apps.

Examples:

- pyargv="one two three" will set `sys.argv` to `('one', 'two', 'three')`.

Parameters **args** –

set_wsgi_params(**, module: Union[str, pathlib.Path] = None, callable_name: str = None, env_strategy: str = None*)

Set wsgi related parameters.

Parameters

- **module** –
  - load .wsgi file as the Python application
  - load a WSGI module as the application.

  Note: The module (sans `.py`) must be importable, ie. be in `PYTHONPATH`.

  Examples:
  - mypackage.my_wsgi_module – read from *application* attr of mypackage/my_wsgi_module.py
  - mypackage.my_wsgi_module:my_app – read from *my_app* attr of mypackage/my_wsgi_module.py
- **callable_name** – Set WSGI callable name. Default: application.
- **env_strategy** – Strategy for allocating/deallocating the WSGI env, can be:
  - **cheat - preallocates the env dictionary on uWSGI startup and clears it** after each request. Default behaviour for uWSGI <= 2.0.x

- **holy - creates and destroys the environ dictionary at each request.**
  Default behaviour for uWSGI >= 2.1

eval_wsgi_entrypoint(*code: str*)
Evaluates Python code as WSGI entry point.

Parameters **code** –

set_autoreload_params(*, *scan_interval: int = None*, *ignore_modules: Union[str, List[str]] = None*)
Sets autoreload related parameters.

Parameters

- **scan_interval** – Seconds. Monitor Python modules' modification times to trigger reload.

  Warning: Use only in development.

- **ignore_modules** – Ignore the specified module during auto-reload scan.

register_module_alias(*alias: str*, *module_path: str*, *, *after_init: bool = False*)
Adds an alias for a module.

http://uwsgi-docs.readthedocs.io/en/latest/PythonModuleAlias.html

Parameters

- **alias** –
- **module_path** –
- **after_init** – add a python module alias after uwsgi module initialization

import_module(*modules: Union[str, int], *, shared: bool = False, into_spooler: bool = False*)
Imports a python module.

Parameters

- **modules** –
- **shared** – If shared import is done once in master process. Otherwise import a python module in all of the processes. This is done after fork but before request processing.
- **into_spooler** – Import a python module in the spooler. http://uwsgi-docs.readthedocs.io/en/latest/Spooler.html

run_module(*module: str*)
Runs a Python script in the uWSGI environment.

Parameters **module** –

3.8.19 Config Formatters

Here belong tools for configuration fomatting purposes.

uwsgiconf.formatters.**format_print_text**(*text: str*, *, *color_fg: str = None*, *color_bg: str = None*) → str
Format given text using ANSI formatting escape sequences.

Could be useful for print command.

Parameters

- **text** –
- **color_fg** – text (foreground) color
- **color_bg** – text (background) color

class uwsgiconf.formatters.**FormatterBase**(*sections: List[Section]*)
Base class for configuration formatters.

iter_options() → Generator[Tuple[str, str, Any], None, None]
Iterates configuration sections groups options.

class uwsgiconf.formatters.**IniFormatter**(*sections: List[Section]*)
Translates a configuration as INI file.

class uwsgiconf.formatters.**ArgsFormatter**(*sections: List[Section]*)
Translates a configuration to command line arguments.

uwsgiconf.formatters.**FORMATTERS = {'args': <class 'uwsgiconf.formatters.ArgsFormatter'>,**
Available formatters by alias.

CHAPTER 4

Get involved into uwsgiconf

Submit issues. If you spotted something weird in application behavior or want to propose a feature you can do that at https://github.com/idlesign/uwsgiconf/issues

Write code. If you are eager to participate in application development, fork it at https://github.com/idlesign/uwsgiconf, write your code, whether it should be a bugfix or a feature implementation, and make a pull request right from the forked project page.

Spread the word. If you have some tips and tricks or any other words in mind that you think might be of interest for the others — publish it.

Python Module Index

u

uwsgiconf.config, 50
uwsgiconf.formatters, 148
uwsgiconf.options.alarm_types, 56
uwsgiconf.options.alarms, 56
uwsgiconf.options.applications, 57
uwsgiconf.options.caching, 59
uwsgiconf.options.empire, 61
uwsgiconf.options.locks, 64
uwsgiconf.options.logging, 68
uwsgiconf.options.logging_encoders, 67
uwsgiconf.options.logging_loggers, 65
uwsgiconf.options.main_process, 76
uwsgiconf.options.main_process_actions, 74
uwsgiconf.options.master_process, 82
uwsgiconf.options.monitoring, 92
uwsgiconf.options.monitoring_collectors, 91
uwsgiconf.options.monitoring_metric_types, 86
uwsgiconf.options.monitoring_pushers, 89
uwsgiconf.options.networking, 100
uwsgiconf.options.networking_sockets, 96
uwsgiconf.options.python, 146
uwsgiconf.options.queue, 104
uwsgiconf.options.routing, 124
uwsgiconf.options.routing_actions, 104
uwsgiconf.options.routing_modifiers, 118
uwsgiconf.options.routing_routers, 109
uwsgiconf.options.routing_subjects, 122
uwsgiconf.options.routing_vars, 123
uwsgiconf.options.spooler, 132
uwsgiconf.options.statics, 133
uwsgiconf.options.subscriptions, 136
uwsgiconf.options.subscriptions_algos, 136
uwsgiconf.options.workers, 141
uwsgiconf.options.workers_cheapening, 139
uwsgiconf.presets.empire, 9
uwsgiconf.presets.nice, 10
uwsgiconf.runtime.alarms, 32
uwsgiconf.runtime.asynced, 32
uwsgiconf.runtime.caching, 33
uwsgiconf.runtime.control, 34
uwsgiconf.runtime.locking, 35
uwsgiconf.runtime.logging, 36
uwsgiconf.runtime.monitoring, 36
uwsgiconf.runtime.mules, 38
uwsgiconf.runtime.rpc, 41
uwsgiconf.runtime.scheduling, 42
uwsgiconf.runtime.signals, 45
uwsgiconf.runtime.spooler, 47
uwsgiconf.uwsgi_stub, 14

Index

Symbols

_Platform (*class in uwsgiconf.runtime.platform*), 39
_Request (*class in uwsgiconf.runtime.request*), 41

A

absolute (*uwsgiconf.options.monitoring.Monitoring.metric_types attribute*), 93
ACCEL_REDIRECT (*uwsgiconf.options.statics.Statics.transfer_modes attribute*), 134
accepting() (*in module uwsgiconf.uwsgi_stub*), 16
accumulator (*uwsgiconf.options.monitoring.Monitoring.collectors attribute*), 93
acquire() (*uwsgiconf.runtime.locking.Lock method*), 36
ActionAddVarCgi (*class in uwsgiconf.options.routing_actions*), 105
ActionAddVarLog (*class in uwsgiconf.options.routing_actions*), 105
ActionAlarm (*class in uwsgiconf.options.main_process_actions*), 75
ActionAlarm (*class in uwsgiconf.options.routing_actions*), 107
ActionAuthBasic (*class in uwsgiconf.options.routing_actions*), 107
ActionCall (*class in uwsgiconf.options.main_process_actions*), 75
ActionChunked (*class in uwsgiconf.options.routing_actions*), 105
ActionDirChange (*class in uwsgiconf.options.main_process_actions*), 75
ActionDirChange (*class in uwsgiconf.options.routing_actions*), 108
ActionDirCreate (*class in uwsgiconf.options.main_process_actions*), 75
ActionDoBreak (*class in uwsgiconf.options.routing_actions*), 105
ActionDoContinue (*class in uwsgiconf.options.routing_actions*), 105
ActionDoGoto (*class in uwsgiconf.options.routing_actions*), 105
ActionExecute (*class in uwsgiconf.options.main_process_actions*), 75
ActionExit (*class in uwsgiconf.options.main_process_actions*), 75
ActionFifoWrite (*class in uwsgiconf.options.main_process_actions*), 76
ActionFileCreate (*class in uwsgiconf.options.main_process_actions*), 75
ActionFileWrite (*class in uwsgiconf.options.main_process_actions*), 75
ActionFixContentLen (*class in uwsgiconf.options.routing_actions*), 105
ActionFixVarPathInfo (*class in uwsgiconf.options.routing_actions*), 109
ActionFlush (*class in uwsgiconf.options.routing_actions*), 104
ActionGzip (*class in uwsgiconf.options.routing_actions*), 105
ActionHeaderAdd (*class in uwsgiconf.options.routing_actions*), 106
ActionHeaderRemove (*class in uwsgiconf.options.routing_actions*), 106
ActionHeadersOff (*class in uwsgiconf.options.routing_actions*), 106
ActionHeadersReset (*class in uwsgiconf.options.routing_actions*), 106
ActionLog (*class in uwsgiconf.options.routing_actions*), 105
ActionMount (*class in uwsgiconf.options.main_process_actions*), 74
ActionOffloadOff (*class in uwsgiconf.options.routing_actions*), 105
ActionPrintout (*class in uwsgiconf.options.main_process_actions*), 75
ActionRedirect (*class in uwsgiconf.options.routing_actions*), 106
ActionRewrite (*class in uwsgi-*

conf.options.routing_actions), 106
ActionRouteExternal (class in uwsgiconf.options.routing_actions), 106
ActionRouteUwsgi (class in uwsgiconf.options.routing_actions), 106
ActionSend (class in uwsgiconf.options.routing_actions), 106
ActionServeStatic (class in uwsgiconf.options.routing_actions), 107
ActionSetHarakiri (class in uwsgiconf.options.routing_actions), 108
ActionSetHostName (class in uwsgiconf.options.main_process_actions), 75
ActionSetScriptFile (class in uwsgiconf.options.routing_actions), 109
ActionSetUwsgiProcessName (class in uwsgiconf.options.routing_actions), 109
ActionSetVarDocumentRoot (class in uwsgiconf.options.routing_actions), 108
ActionSetVarPathInfo (class in uwsgiconf.options.routing_actions), 108
ActionSetVarRemoteAddr (class in uwsgiconf.options.routing_actions), 108
ActionSetVarRemoteUser (class in uwsgiconf.options.routing_actions), 108
ActionSetVarRequestMethod (class in uwsgiconf.options.routing_actions), 108
ActionSetVarRequestUri (class in uwsgiconf.options.routing_actions), 108
ActionSetVarScriptName (class in uwsgiconf.options.routing_actions), 108
ActionSetVarUwsgiAppid (class in uwsgiconf.options.routing_actions), 108
ActionSetVarUwsgiHome (class in uwsgiconf.options.routing_actions), 108
ActionSetVarUwsgiScheme (class in uwsgiconf.options.routing_actions), 108
ActionSignal (class in uwsgiconf.options.routing_actions), 106
ActionTemplate (class in uwsgiconf.options.routing_actions), 105
ActionToFile (class in uwsgiconf.options.routing_actions), 105
ActionUnlink (class in uwsgiconf.options.main_process_actions), 76
ActionUpper (class in uwsgiconf.options.routing_actions), 105
add() (uwsgiconf.options.spooler.Spooler method), 133
add_cache() (uwsgiconf.options.caching.Caching method), 60
add_cron() (in module uwsgiconf.uwsgi_stub), 16
add_cron_task() (uwsgiconf.options.master_process.MasterProcess method), 83
add_expiration_rule() (uwsgiconf.options.statics.Statics method), 135
add_file() (uwsgiconf.options.caching.Caching method), 59
add_file_monitor() (in module uwsgiconf.uwsgi_stub), 16
add_firewall_rule() (uwsgiconf.options.routing_routers.RouterTunTap method), 118
add_item() (uwsgiconf.options.caching.Caching method), 59
add_logger() (uwsgiconf.options.logging.Logging method), 71
add_logger_encoder() (uwsgiconf.options.logging.Logging method), 71
add_logger_route() (uwsgiconf.options.logging.Logging method), 71
add_ms_timer() (in module uwsgiconf.uwsgi_stub), 16
add_rb_timer() (in module uwsgiconf.uwsgi_stub), 16
add_timer() (in module uwsgiconf.uwsgi_stub), 16
add_var() (in module uwsgiconf.uwsgi_stub), 17
add_var() (uwsgiconf.runtime.request._Request method), 41
add_var_cgi (uwsgiconf.options.routing.RouteRule.actions attribute), 127
add_var_log (uwsgiconf.options.routing.RouteRule.actions attribute), 127
adder (uwsgiconf.options.monitoring.Monitoring.collectors attribute), 93
alarm (uwsgiconf.options.main_process.MainProcess.actions attribute), 76
alarm (uwsgiconf.options.routing.RouteRule.actions attribute), 127
alarm() (in module uwsgiconf.uwsgi_stub), 17
alarm_on_fd_ready() (uwsgiconf.options.alarms.Alarms method), 57
alarm_on_log() (uwsgiconf.options.alarms.Alarms method), 57
alarm_on_queue_full() (uwsgiconf.options.alarms.Alarms method), 57
alarm_on_segfault() (uwsgiconf.options.alarms.Alarms method), 57
AlarmCommand (class in uwsgiconf.options.alarm_types), 56
AlarmCurl (class in uwsgiconf.options.alarm_types), 56
AlarmLog (class in uwsgiconf.options.alarm_types), 56
AlarmMule (class in uwsgiconf.options.alarm_types), 56
Alarms (class in uwsgiconf.options.alarms), 56

alarms (*uwsgiconf.config.Section attribute*), 51
Alarms.alarm_types (*class in uwsgiconf.options.alarms*), 56
AlarmSignal (*class in uwsgiconf.options.alarm_types*), 56
AlarmType (*class in uwsgiconf.options.alarm_types*), 56
AlarmXmpp (*class in uwsgiconf.options.alarm_types*), 56
Algo (*class in uwsgiconf.options.workers_cheapening*), 139
AlgoBusyness (*class in uwsgiconf.options.workers_cheapening*), 139
AlgoManual (*class in uwsgiconf.options.workers_cheapening*), 140
AlgoQueue (*class in uwsgiconf.options.workers_cheapening*), 139
AlgoSpare (*class in uwsgiconf.options.workers_cheapening*), 139
AlgoSpare2 (*class in uwsgiconf.options.workers_cheapening*), 139
alias (*uwsgiconf.options.monitoring.Monitoring.metric_types attribute*), 93
APP_LOAD_POST (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
APP_LOAD_PRE (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
Applications (*class in uwsgiconf.options.applications*), 57
applications (*in module uwsgiconf.uwsgi_stub*), 15
applications (*uwsgiconf.config.Section attribute*), 51
apps_map (*uwsgiconf.runtime.platform._Platform attribute*), 40
ArgsFormatter (*class in uwsgiconf.formatters*), 149
as_configuration() (*uwsgiconf.config.Section method*), 52
ASAP (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 77
async_connect() (*in module uwsgiconf.uwsgi_stub*), 17
async_sleep() (*in module uwsgiconf.uwsgi_stub*), 17
ASYNC_SWITCHES (*uwsgiconf.options.logging.Logging.vars attribute*), 74
attach_process() (*uwsgiconf.options.master_process.MasterProcess method*), 85
attach_process_classic() (*uwsgiconf.options.master_process.MasterProcess method*), 84
auth_basic (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
auth_ldap (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
AuthLdap (*class in uwsgiconf.options.routing_actions*), 107
avg (*uwsgiconf.options.monitoring.Monitoring.collectors attribute*), 93

B

BalancingAlgorithm (*class in uwsgiconf.options.subscriptions_algos*), 136
BalancingAlgorithmWithBackup (*class in uwsgiconf.options.subscriptions_algos*), 136
base64 (*uwsgiconf.options.routing.RouteRule.var_functions attribute*), 125
BASIC (*uwsgiconf.config.Section.embedded_plugins_presets attribute*), 52
bootstrap() (*uwsgiconf.config.Section class method*), 54
Broodlord (*class in uwsgiconf.presets.empire*), 9
buffer_size (*in module uwsgiconf.uwsgi_stub*), 15
buffer_size (*uwsgiconf.runtime.platform._Platform attribute*), 40
busyness (*uwsgiconf.options.workers_cheapening.Cheapening.algorithm attribute*), 140

C

Cache (*class in uwsgiconf.runtime.caching*), 33
cache (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
cache_clear() (*in module uwsgiconf.uwsgi_stub*), 17
cache_dec() (*in module uwsgiconf.uwsgi_stub*), 17
cache_del() (*in module uwsgiconf.uwsgi_stub*), 17
cache_div() (*in module uwsgiconf.uwsgi_stub*), 18
cache_exists() (*in module uwsgiconf.uwsgi_stub*), 18
cache_get() (*in module uwsgiconf.uwsgi_stub*), 18
cache_inc() (*in module uwsgiconf.uwsgi_stub*), 18
cache_keys() (*in module uwsgiconf.uwsgi_stub*), 18
cache_mul() (*in module uwsgiconf.uwsgi_stub*), 18
cache_num() (*in module uwsgiconf.uwsgi_stub*), 19
cache_set() (*in module uwsgiconf.uwsgi_stub*), 19
cache_update() (*in module uwsgiconf.uwsgi_stub*), 19
Caching (*class in uwsgiconf.options.caching*), 59
caching (*uwsgiconf.config.Section attribute*), 51
call (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 76
call() (*in module uwsgiconf.uwsgi_stub*), 19
carbon (*uwsgiconf.options.monitoring.Monitoring.pushers attribute*), 93

cgi (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
change_dir() (*uwsgiconf.options.main_process.MainProcess method*), 80
Cheapening (*class in uwsgiconf.options.workers_cheapening*), 140
cheapening (*uwsgiconf.config.Section attribute*), 51
Cheapening.algorithms (*class in uwsgiconf.options.workers_cheapening*), 140
chunked (*uwsgiconf.options.routing.RouteRule.transforms attribute*), 126
chunked_read() (*in module uwsgiconf.uwsgi_stub*), 19
chunked_read_nb() (*in module uwsgiconf.uwsgi_stub*), 19
cl() (*in module uwsgiconf.uwsgi_stub*), 19
clear() (*uwsgiconf.runtime.caching.Cache method*), 33
clock (*uwsgiconf.runtime.platform._Platform attribute*), 40
close() (*in module uwsgiconf.uwsgi_stub*), 20
cluster_node (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
Collector (*class in uwsgiconf.options.monitoring_collectors*), 91
CollectorAccumulator (*class in uwsgiconf.options.monitoring_collectors*), 92
CollectorAdder (*class in uwsgiconf.options.monitoring_collectors*), 92
CollectorAvg (*class in uwsgiconf.options.monitoring_collectors*), 92
CollectorFile (*class in uwsgiconf.options.monitoring_collectors*), 91
CollectorFunction (*class in uwsgiconf.options.monitoring_collectors*), 92
CollectorMultiplier (*class in uwsgiconf.options.monitoring_collectors*), 92
CollectorPointer (*class in uwsgiconf.options.monitoring_collectors*), 91
CollectorSum (*class in uwsgiconf.options.monitoring_collectors*), 92
command (*uwsgiconf.options.alarms.Alarms.alarm_types attribute*), 56
compress (*uwsgiconf.options.logging.Logging.encoders attribute*), 68
CONF_CURRENT_SECTION (*uwsgiconf.config.Section.vars attribute*), 54
CONF_NAME_ORIGINAL (*uwsgiconf.config.Section.vars attribute*), 54
config (*uwsgiconf.runtime.platform._Platform attribute*), 40
config_from_node (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
config_variables (*uwsgiconf.runtime.platform._Platform attribute*), 40
Configuration (*class in uwsgiconf.config*), 55
configure() (*uwsgiconf.presets.empire.Broodlord method*), 10
configure_certbot_https() (*uwsgiconf.presets.nice.Section method*), 11
configure_https_redirect() (*uwsgiconf.presets.nice.Section method*), 11
configure_logging_json() (*uwsgiconf.presets.nice.Section method*), 12
configure_maintenance_mode() (*uwsgiconf.presets.nice.Section method*), 11
configure_owner() (*uwsgiconf.presets.nice.Section method*), 11
configure_uwsgi() (*in module uwsgiconf.config*), 55
connect() (*in module uwsgiconf.uwsgi_stub*), 20
connection_fd() (*in module uwsgiconf.uwsgi_stub*), 20
contains() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
contains_ipv4() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
contains_ipv6() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
content_length (*uwsgiconf.runtime.request._Request attribute*), 41
cookie (*uwsgiconf.options.routing.RouteRule.vars attribute*), 124
CORE (*uwsgiconf.options.logging.Logging.vars attribute*), 74
corerouter_signal (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
cores (*in module uwsgiconf.uwsgi_stub*), 15
cores_count (*uwsgiconf.runtime.platform._Platform attribute*), 40
counter (*uwsgiconf.options.monitoring.Monitoring.metric_types attribute*), 93
CPU_CORES (*uwsgiconf.config.Section.vars attribute*), 54
curl (*uwsgiconf.options.alarms.Alarms.alarm_types attribute*), 56
custom (*uwsgiconf.options.routing.RouteRule.subjects attribute*), 125

D

daemonize() (*uwsgiconf.options.main_process.MainProcess method*), 79
decr() (*uwsgiconf.runtime.caching.Cache method*), 34
decr() (*uwsgiconf.runtime.monitoring.Metric method*), 37
default (*uwsgiconf.options.networking.Networking.sockets attribute*), 100
delete() (*uwsgiconf.runtime.caching.Cache method*), 34
derive_from() (*uwsgiconf.config.Section class method*), 53
device_add_rule() (*uwsgiconf.options.routing_routers.RouterTunTap method*), 117
device_connect() (*uwsgiconf.options.routing_routers.RouterTunTap method*), 117
dir_change (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 76
dir_change (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
dir_create (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 76
DIR_DOCUMENT_ROOT (*uwsgiconf.options.statics.Statics attribute*), 133
DIR_VASSALS (*uwsgiconf.config.Section.vars attribute*), 54
disconnect() (*in module uwsgiconf.uwsgi_stub*), 20
div() (*uwsgiconf.runtime.caching.Cache method*), 34
div() (*uwsgiconf.runtime.monitoring.Metric method*), 37
do_break (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
do_continue (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
do_goto (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127

E

echo (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
embedded_data() (*in module uwsgiconf.uwsgi_stub*), 20
EMPEROR_LOST (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 77
EMPEROR_RELOAD (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 77
EMPEROR_START (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 77
EMPEROR_STOP (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 77
Empire (*class in uwsgiconf.options.empire*), 61
empire (*uwsgiconf.config.Section attribute*), 51
enable() (*uwsgiconf.options.queue.Queue method*), 104
enable_snmp() (*uwsgiconf.options.monitoring.Monitoring method*), 95
Encoder (*class in uwsgiconf.options.logging_encoders*), 67
EncoderCompress (*class in uwsgiconf.options.logging_encoders*), 67
EncoderFormat (*class in uwsgiconf.options.logging_encoders*), 67
EncoderFormat.vars (*class in uwsgiconf.options.logging_encoders*), 67
EncoderGzip (*class in uwsgiconf.options.logging_encoders*), 67
EncoderJson (*class in uwsgiconf.options.logging_encoders*), 67
EncoderNewline (*class in uwsgiconf.options.logging_encoders*), 67
EncoderPrefix (*class in uwsgiconf.options.logging_encoders*), 67
EncoderSuffix (*class in uwsgiconf.options.logging_encoders*), 67
endswith() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
env (*in module uwsgiconf.uwsgi_stub*), 15
env (*uwsgiconf.runtime.request._Request attribute*), 41
env() (*uwsgiconf.config.Section method*), 53
eq() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
ERROR (*uwsgiconf.options.routing.RouteRule.stages attribute*), 125
eval (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
eval_wsgi_entrypoint() (*uwsgiconf.options.python.Python method*), 148
example (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
execute (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 76
exists() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
exit (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 76
EXIT (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
extract() (*in module uwsgiconf.uwsgi_stub*), 20

F

Farm (*class in uwsgiconf.runtime.mules*), 38
farm_get_msg() (*in module uwsgiconf.uwsgi_stub*), 20
farm_msg() (*in module uwsgiconf.uwsgi_stub*), 20
fast (*uwsgiconf.options.routing.Routing.routers attribute*), 129
fastcgi (*uwsgiconf.options.networking.Networking.sockets attribute*), 100
fastfunc (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
fd (*uwsgiconf.options.logging.Logging.loggers attribute*), 68
fd (*uwsgiconf.runtime.request._Request attribute*), 41
fifo_write (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 76
file (*uwsgiconf.options.logging.Logging.loggers attribute*), 68
file (*uwsgiconf.options.monitoring.Monitoring.collectors attribute*), 93
file (*uwsgiconf.options.monitoring.Monitoring.pushers attribute*), 93
file_create (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 76
file_write (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 77
FILENAME (*uwsgiconf.options.statics.Statics.expiration_criteria attribute*), 134
FINAL (*uwsgiconf.options.routing.RouteRule.stages attribute*), 125
fix_content_len (*uwsgiconf.options.routing.RouteRule.transforms attribute*), 126
fix_var_path_info (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
flush (*uwsgiconf.options.routing.RouteRule.transforms attribute*), 126
forkpty (*uwsgiconf.options.routing.Routing.routers attribute*), 129
format (*uwsgiconf.options.logging.Logging.encoders attribute*), 68
format() (*uwsgiconf.config.Configuration method*), 55
FORMAT_ESCAPE (*uwsgiconf.config.Section.vars attribute*), 54
format_print_text() (*in module uwsgiconf.formatters*), 148
FormatterBase (*class in uwsgiconf.formatters*), 149
FORMATTERS (*in module uwsgiconf.formatters*), 149
Forwarder (*class in uwsgiconf.options.routing_routers*), 109
ForwarderCache (*class in uwsgiconf.options.routing_routers*), 109
ForwarderCode (*class in uwsgiconf.options.routing_routers*), 109
ForwarderPath (*class in uwsgiconf.options.routing_routers*), 109
ForwarderSocket (*class in uwsgiconf.options.routing_routers*), 110
ForwarderSubscriptionServer (*class in uwsgiconf.options.routing_routers*), 110
from_dsn() (*uwsgiconf.options.networking.Networking.sockets class method*), 101
Func (*class in uwsgiconf.options.routing_vars*), 123
func (*uwsgiconf.runtime.signals.SignalDescription attribute*), 45
FuncBase64 (*class in uwsgiconf.options.routing_vars*), 124
FuncHex (*class in uwsgiconf.options.routing_vars*), 124
FuncLower (*class in uwsgiconf.options.routing_vars*), 124
FuncMath (*class in uwsgiconf.options.routing_vars*), 124
FuncMime (*class in uwsgiconf.options.routing_vars*), 124
function (*uwsgiconf.options.monitoring.Monitoring.collectors attribute*), 93
FuncUpper (*class in uwsgiconf.options.routing_vars*), 124

G

GATEWAY_START_IN_EACH (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
gauge (*uwsgiconf.options.monitoring.Monitoring.metric_types attribute*), 93
ge() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
geoip (*uwsgiconf.options.routing.RouteRule.vars attribute*), 124
get() (*uwsgiconf.runtime.caching.Cache method*), 33
get_available_num() (*in module uwsgiconf.runtime.signals*), 45
get_bundled_static_path() (*uwsgiconf.presets.nice.Section class method*), 11
get_by_basename() (*uwsgiconf.runtime.spooler.Spooler class method*), 48
get_certbot_paths() (*uwsgiconf.options.networking_sockets.SocketHttps class method*), 97
get_current() (*uwsgiconf.runtime.mules.Mule class method*), 38
get_current_id() (*uwsgiconf.runtime.mules.Mule class method*), 38

get_descriptions() (*uwsgiconf.config.Section.vars class method*), 54
get_farms() (*uwsgiconf.runtime.mules.Farm class method*), 39
get_last_received() (*in module uwsgiconf.runtime.signals*), 45
get_listen_queue() (*uwsgiconf.runtime.platform._Platform method*), 40
get_log_format_default() (*uwsgiconf.presets.nice.Section method*), 10
get_logvar() (*in module uwsgiconf.uwsgi_stub*), 20
get_message() (*uwsgiconf.runtime.mules.Farm class method*), 39
get_message() (*uwsgiconf.runtime.mules.Mule class method*), 38
get_owner() (*uwsgiconf.options.main_process.MainProcess method*), 80
get_pids() (*uwsgiconf.runtime.spooler.Spooler class method*), 48
get_rpc_list() (*in module uwsgiconf.runtime.rpc*), 42
get_runtime_dir() (*uwsgiconf.config.Section method*), 52
get_spoolers() (*uwsgiconf.runtime.spooler.Spooler class method*), 48
get_tasks() (*uwsgiconf.runtime.spooler.Spooler class method*), 48
get_version() (*uwsgiconf.runtime.platform._Platform method*), 41
glusterfs (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
green_schedule() (*in module uwsgiconf.uwsgi_stub*), 20
gridfs (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
GROUP_ID (*uwsgiconf.config.Section.vars attribute*), 54
GROUP_NAME (*uwsgiconf.config.Section.vars attribute*), 54
gt() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
gzip (*uwsgiconf.options.logging.Logging.encoders attribute*), 68
gzip (*uwsgiconf.options.routing.RouteRule.transforms attribute*), 126

H

harakiri_imposed (*class in uwsgiconf.runtime.control*), 34
has_threads (*in module uwsgiconf.uwsgi_stub*), 15
header_add (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
header_add() (*uwsgiconf.options.routing.Routing method*), 132
header_collect() (*uwsgiconf.options.routing.Routing method*), 132
header_remove (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
header_remove() (*uwsgiconf.options.routing.Routing method*), 132
headers_off (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
headers_reset (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
hex (*uwsgiconf.options.routing.RouteRule.var_functions attribute*), 125
HookAction (*class in uwsgiconf.options.main_process_actions*), 74
HOST_NAME (*uwsgiconf.config.Section.vars attribute*), 54
hostname (*in module uwsgiconf.uwsgi_stub*), 15
hostname (*uwsgiconf.runtime.platform._Platform attribute*), 40
http (*uwsgiconf.options.networking.Networking.sockets attribute*), 101
http (*uwsgiconf.options.routing.Routing.routers attribute*), 129
http_host (*uwsgiconf.options.routing.RouteRule.subjects attribute*), 126
http_referer (*uwsgiconf.options.routing.RouteRule.subjects attribute*), 126
http_user_agent (*uwsgiconf.options.routing.RouteRule.subjects attribute*), 126
https (*uwsgiconf.options.networking.Networking.sockets attribute*), 101
https (*uwsgiconf.options.routing.Routing.routers attribute*), 129
httptime (*uwsgiconf.options.routing.RouteRule.vars attribute*), 125

I

i_am_the_lord() (*in module uwsgiconf.uwsgi_stub*), 29
i_am_the_spooler() (*in module uwsgiconf.uwsgi_stub*), 20
id (*uwsgiconf.runtime.request._Request attribute*), 41
import_module() (*uwsgiconf.options.python.Python method*), 148
in_farm() (*in module uwsgiconf.uwsgi_stub*), 20
include() (*uwsgiconf.config.Section method*), 53
incr() (*uwsgiconf.runtime.caching.Cache method*), 34

incr() (*uwsgiconf.runtime.monitoring.Metric method*), 37
IniFormatter (*class in uwsgiconf.formatters*), 149
ip_hash (*uwsgiconf.options.subscriptions.Subscriptions.algorithms attribute*), 136
IpHash (*class in uwsgiconf.options.subscriptions_algos*), 136
is_a_reload() (*in module uwsgiconf.uwsgi_stub*), 21
is_connected() (*in module uwsgiconf.uwsgi_stub*), 21
is_locked() (*in module uwsgiconf.uwsgi_stub*), 21
is_mine (*uwsgiconf.runtime.mules.Farm attribute*), 39
is_set (*uwsgiconf.runtime.locking.Lock attribute*), 35
is_stub (*in module uwsgiconf.uwsgi_stub*), 14
isdir() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
isempty() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
isexec() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
isfile() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
islink() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
islord() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
iter_options() (*uwsgiconf.formatters.FormatterBase method*), 149

J

JAIL_IN (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 77
JAIL_POST (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 77
JAIL_PRE (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 77
json (*uwsgiconf.options.logging.Logging.encoders attribute*), 68
jvm (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129

K

keys (*uwsgiconf.runtime.caching.Cache attribute*), 33

L

le() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
least_reference_count (*uwsgiconf.options.subscriptions.Subscriptions.algorithms attribute*), 136
LeastReferenceCount (*class in uwsgiconf.options.subscriptions_algos*), 136
legion_msg (*uwsgiconf.options.routing.Routing.modifiers attribute*), 129
listen_queue() (*in module uwsgiconf.uwsgi_stub*), 21
Lock (*class in uwsgiconf.runtime.locking*), 35
lock (*in module uwsgiconf.runtime.locking*), 36
lock() (*in module uwsgiconf.uwsgi_stub*), 21
lock_file() (*uwsgiconf.options.locks.Locks method*), 64
Locks (*class in uwsgiconf.options.locks*), 64
locks (*uwsgiconf.config.Section attribute*), 51
log (*uwsgiconf.options.alarms.Alarms.alarm_types attribute*), 56
log (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
log() (*in module uwsgiconf.uwsgi_stub*), 21
log() (*uwsgiconf.runtime.request._Request method*), 41
log_into() (*uwsgiconf.options.logging.Logging method*), 69
log_this_request() (*in module uwsgiconf.uwsgi_stub*), 21
Logger (*class in uwsgiconf.options.logging_loggers*), 65
LoggerFile (*class in uwsgiconf.options.logging_loggers*), 65
LoggerFileDescriptor (*class in uwsgiconf.options.logging_loggers*), 65
LoggerMongo (*class in uwsgiconf.options.logging_loggers*), 66
LoggerRedis (*class in uwsgiconf.options.logging_loggers*), 66
LoggerRsyslog (*class in uwsgiconf.options.logging_loggers*), 65
LoggerSocket (*class in uwsgiconf.options.logging_loggers*), 65
LoggerStdIO (*class in uwsgiconf.options.logging_loggers*), 65
LoggerSyslog (*class in uwsgiconf.options.logging_loggers*), 65
LoggerZeroMq (*class in uwsgiconf.options.logging_loggers*), 66
Logging (*class in uwsgiconf.options.logging*), 68
logging (*uwsgiconf.config.Section attribute*), 51
Logging.encoders (*class in uwsgiconf.options.logging*), 68
Logging.loggers (*class in uwsgiconf.options.logging*), 68
Logging.vars (*class in uwsgiconf.options.logging*), 72
logsize() (*in module uwsgiconf.uwsgi_stub*), 21
loop() (*in module uwsgiconf.uwsgi_stub*), 21
lord_scroll() (*in module uwsgiconf.uwsgi_stub*), 29
lower (*uwsgiconf.options.routing.RouteRule.var_functions*

attribute), 125
lt() (uwsgiconf.options.routing_subjects.SubjectCustom method), 122
lua (uwsgiconf.options.routing.Routing.modifiers attribute), 130

M

magic_table (in module uwsgiconf.uwsgi_stub), 15
main_process (uwsgiconf.config.Section attribute), 51
MainProcess (class in uwsgiconf.options.main_process), 76
MainProcess.actions (class in uwsgiconf.options.main_process), 76
MainProcess.phases (class in uwsgiconf.options.main_process), 77
make_rpc_call() (in module uwsgiconf.runtime.rpc), 42
manage (uwsgiconf.options.routing.Routing.modifiers attribute), 130
manage_path_info (uwsgiconf.options.routing.Routing.modifiers attribute), 130
manual (uwsgiconf.options.workers_cheapening.Cheapening.algorithms attribute), 140
mark_processed (uwsgiconf.runtime.spooler.SpoolerTask attribute), 49
mark_rescheduled (uwsgiconf.runtime.spooler.SpoolerTask attribute), 49
mark_skipped (uwsgiconf.runtime.spooler.SpoolerTask attribute), 49
master_pid (uwsgiconf.runtime.platform._Platform attribute), 40
master_process (uwsgiconf.config.Section attribute), 51
MASTER_START (uwsgiconf.options.main_process.MainProcess.phases attribute), 77
masterpid() (in module uwsgiconf.uwsgi_stub), 21
MasterProcess (class in uwsgiconf.options.master_process), 82
matches() (uwsgiconf.options.routing_subjects.SubjectCustom method), 122
math (uwsgiconf.options.routing.RouteRule.var_functions attribute), 125
mem() (in module uwsgiconf.uwsgi_stub), 21
MEM_RSS (uwsgiconf.options.logging.Logging.vars attribute), 74
MEM_RSS_MV (uwsgiconf.options.logging.Logging.vars attribute), 74
MEM_VSZ (uwsgiconf.options.logging.Logging.vars attribute), 74
MEM_VSZ_MB (uwsgiconf.options.logging.Logging.vars attribute), 74
memory (uwsgiconf.runtime.platform._Platform attribute), 40
MESSAGE (uwsgiconf.options.logging_encoders.EncoderFormat.vars attribute), 67
message (uwsgiconf.options.routing.Routing.modifiers attribute), 130
message_array (uwsgiconf.options.routing.Routing.modifiers attribute), 130
message_marshal (uwsgiconf.options.routing.Routing.modifiers attribute), 130
MESSAGE_NEWLINE (uwsgiconf.options.logging_encoders.EncoderFormat.vars attribute), 67
Metric (class in uwsgiconf.options.monitoring_metric_types), 86
Metric (class in uwsgiconf.runtime.monitoring), 36
metric (uwsgiconf.options.logging.Logging.vars attribute), 74
metric (uwsgiconf.options.routing.RouteRule.vars attribute), 125
metric_dec() (in module uwsgiconf.uwsgi_stub), 21
metric_div() (in module uwsgiconf.uwsgi_stub), 22
metric_get() (in module uwsgiconf.uwsgi_stub), 22
metric_inc() (in module uwsgiconf.uwsgi_stub), 22
metric_mul() (in module uwsgiconf.uwsgi_stub), 22
metric_set() (in module uwsgiconf.uwsgi_stub), 22
metric_set_max() (in module uwsgiconf.uwsgi_stub), 22
metric_set_min() (in module uwsgiconf.uwsgi_stub), 22
MetricTypeAbsolute (class in uwsgiconf.options.monitoring_metric_types), 88
MetricTypeAlias (class in uwsgiconf.options.monitoring_metric_types), 88
MetricTypeCounter (class in uwsgiconf.options.monitoring_metric_types), 86
MetricTypeGauge (class in uwsgiconf.options.monitoring_metric_types), 87
micros() (in module uwsgiconf.uwsgi_stub), 22
mime (uwsgiconf.options.routing.RouteRule.var_functions attribute), 125
MIME_TYPE (uwsgiconf.options.statics.Statics.expiration_criteria attribute), 134
MOD1 (uwsgiconf.options.logging.Logging.vars attribute), 74
MOD2 (uwsgiconf.options.logging.Logging.vars attribute), 74
ModifierCache (class in uwsgi-

conf.options.routing_modifiers), 120
ModifierCgi (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierClusterNode (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierConfigFromNode (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierCorerouterSignal (*class in uwsgiconf.options.routing_modifiers*), 121
ModifierEcho (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierEval (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierExample (*class in uwsgiconf.options.routing_modifiers*), 121
ModifierFastfunc (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierGlusterfs (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierGridfs (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierJvm (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierLegionMsg (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierLua (*class in uwsgiconf.options.routing_modifiers*), 118
ModifierManage (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierManagePathInfo (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierMessage (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierMessageArray (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierMessageMarshal (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierMono (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierMulticast (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierMulticastAnnounce (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierPersistentClose (*class in uwsgiconf.options.routing_modifiers*), 121
ModifierPhp (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierPing (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierPsgi (*class in uwsgiconf.options.routing_modifiers*), 118
ModifierRack (*class in uwsgiconf.options.routing_modifiers*), 118
ModifierRados (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierRaw (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierReload (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierReloadBrutal (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierRemoteLogging (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierResponse (*class in uwsgiconf.options.routing_modifiers*), 121
ModifierRpc (*class in uwsgiconf.options.routing_modifiers*), 121
ModifierSignal (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierSnmp (*class in uwsgiconf.options.routing_modifiers*), 120
ModifierSpooler (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierSubscription (*class in uwsgiconf.options.routing_modifiers*), 121
ModifierSymcall (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierV8 (*class in uwsgiconf.options.routing_modifiers*), 119
ModifierWsgi (*class in uwsgiconf.options.routing_modifiers*), 118
ModifierXslt (*class in uwsgiconf.options.routing_modifiers*), 119
mongo (*uwsgiconf.options.logging.Logging.loggers attribute*), 68
mongo (*uwsgiconf.options.monitoring.Monitoring.pushers attribute*), 93
Monitoring (*class in uwsgiconf.options.monitoring*), 92
monitoring (*uwsgiconf.config.Section attribute*), 51
Monitoring.collectors (*class in uwsgiconf.options.monitoring*), 93
Monitoring.metric_types (*class in uwsgiconf.options.monitoring*), 92
Monitoring.pushers (*class in uwsgiconf.options.monitoring*), 93
mono (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
mount (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 77
mount() (*uwsgiconf.options.applications.Applications method*), 58
mul() (*uwsgiconf.runtime.caching.Cache method*), 34
mul() (*uwsgiconf.runtime.monitoring.Metric method*), 37
Mule (*class in uwsgiconf.runtime.mules*), 38
mule (*uwsgiconf.options.alarms.Alarms.alarm_types attribute*), 56

mule_farm (*uwsgiconf.options.workers.Workers attribute*), 141
mule_get_msg() (*in module uwsgiconf.uwsgi_stub*), 22
mule_id() (*in module uwsgiconf.uwsgi_stub*), 23
mule_msg() (*in module uwsgiconf.uwsgi_stub*), 23
mule_msg_hook() (*in module uwsgiconf.uwsgi_stub*), 15
mule_offload() (*in module uwsgiconf.runtime.mules*), 38
MULE_START_IN_EACH (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
MuleFarm (*class in uwsgiconf.options.workers*), 141
multicast (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
multicast_announce (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
multiplier (*uwsgiconf.options.monitoring.Monitoring.collectors attribute*), 93

N

Networking (*class in uwsgiconf.options.networking*), 100
networking (*uwsgiconf.config.Section attribute*), 51
Networking.sockets (*class in uwsgiconf.options.networking*), 100
newline (*uwsgiconf.options.logging.Logging.encoders attribute*), 68
num (*uwsgiconf.runtime.signals.SignalDescription attribute*), 45
numproc (*in module uwsgiconf.uwsgi_stub*), 15

O

offload() (*in module uwsgiconf.uwsgi_stub*), 23
offload() (*uwsgiconf.runtime.mules.Farm method*), 39
offload() (*uwsgiconf.runtime.mules.Mule method*), 38
offload_off (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
opt (*in module uwsgiconf.uwsgi_stub*), 15

P

parsefile() (*in module uwsgiconf.uwsgi_stub*), 23
path_info (*uwsgiconf.options.routing.RouteRule.subjects attribute*), 126
PATH_INFO (*uwsgiconf.options.statics.Statics.expiration_criteria attribute*), 134
persistent_close (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
php (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
ping (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
pointer (*uwsgiconf.options.monitoring.Monitoring.collectors attribute*), 93
post_fork_hook (*in module uwsgiconf.uwsgi_stub*), 15
postfork_hooks (*uwsgiconf.runtime.platform._Platform attribute*), 40
prefix (*uwsgiconf.options.logging.Logging.encoders attribute*), 68
print_alarms() (*uwsgiconf.options.alarms.Alarms method*), 57
print_alorithms() (*uwsgiconf.options.workers_cheapening.Cheapening method*), 141
print_ini() (*uwsgiconf.config.Configuration method*), 55
print_loggers() (*uwsgiconf.options.logging.Logging method*), 71
print_monitors() (*uwsgiconf.options.empire.Empire method*), 62
print_out() (*uwsgiconf.config.Section method*), 52
print_plugins() (*uwsgiconf.config.Section method*), 52
print_routing_rules() (*uwsgiconf.options.routing.Routing method*), 131
print_stamp() (*uwsgiconf.config.Section method*), 52
print_variables() (*uwsgiconf.config.Section method*), 52
printout (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 77
PRIV_DROP_POST (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 77
PRIV_DROP_PRE (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 77
PROBE() (*uwsgiconf.config.Section.embedded_plugins_presets static method*), 52
process() (*uwsgiconf.runtime.spooler.SpoolerFunctionCallTask method*), 49
process() (*uwsgiconf.runtime.spooler.SpoolerTask method*), 49
project_name (*uwsgiconf.config.Section attribute*), 52
psgi (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
Pusher (*class in uwsgiconf.options.monitoring_pushers*), 89
PusherCarbon (*class in uwsgi-*

conf.options.monitoring_pushers), 90
PusherFile (*class in uwsgiconf.options.monitoring_pushers*), 91
PusherMongo (*class in uwsgiconf.options.monitoring_pushers*), 91
PusherRrdtool (*class in uwsgiconf.options.monitoring_pushers*), 89
PusherSocket (*class in uwsgiconf.options.monitoring_pushers*), 89
PusherStatsd (*class in uwsgiconf.options.monitoring_pushers*), 90
PusherZabbix (*class in uwsgiconf.options.monitoring_pushers*), 91
Python (*class in uwsgiconf.options.python*), 146
python (*uwsgiconf.config.Section attribute*), 51
PythonSection (*class in uwsgiconf.presets.nice*), 12

Q

query (*uwsgiconf.options.routing.RouteRule.vars attribute*), 125
query_string (*uwsgiconf.options.routing.RouteRule.subjects attribute*), 126
Queue (*class in uwsgiconf.options.queue*), 104
queue (*uwsgiconf.config.Section attribute*), 51
queue (*uwsgiconf.options.workers_cheapening.Cheapening.algorithms attribute*), 140

R

rack (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
rados (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
raw (*uwsgiconf.options.networking.Networking.sockets attribute*), 101
raw (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
raw (*uwsgiconf.options.routing.Routing.routers attribute*), 129
read_task_file() (*uwsgiconf.runtime.spooler.Spooler class method*), 48
ready() (*in module uwsgiconf.uwsgi_stub*), 23
ready_fd() (*in module uwsgiconf.uwsgi_stub*), 23
ready_for_requests (*uwsgiconf.runtime.platform._Platform attribute*), 40
recv() (*in module uwsgiconf.uwsgi_stub*), 23
redirect (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
redis (*uwsgiconf.options.logging.Logging.loggers attribute*), 68
register_alarm() (*uwsgiconf.options.alarms.Alarms method*), 57
register_cron() (*in module uwsgiconf.runtime.scheduling*), 44
register_file_monitor() (*in module uwsgiconf.runtime.monitoring*), 36
register_handler() (*uwsgiconf.runtime.signals.Signal method*), 46
register_metric() (*uwsgiconf.options.monitoring.Monitoring method*), 94
register_module_alias() (*uwsgiconf.options.python.Python method*), 148
register_route() (*uwsgiconf.options.routing.Routing method*), 131
register_route() (*uwsgiconf.options.routing_routers.RouterTunTap method*), 117
register_rpc() (*in module uwsgiconf.runtime.rpc*), 41
register_rpc() (*in module uwsgiconf.uwsgi_stub*), 23
register_signal() (*in module uwsgiconf.uwsgi_stub*), 23
register_socket() (*uwsgiconf.options.networking.Networking method*), 102
register_static_map() (*uwsgiconf.options.statics.Statics method*), 135
register_stats_pusher() (*uwsgiconf.options.monitoring.Monitoring method*), 95
register_timer() (*in module uwsgiconf.runtime.scheduling*), 42
register_timer_ms() (*in module uwsgiconf.runtime.scheduling*), 44
register_timer_rb() (*in module uwsgiconf.runtime.scheduling*), 43
registered (*uwsgiconf.runtime.signals.Signal attribute*), 46
registry_signals (*in module uwsgiconf.runtime.signals*), 45
release() (*uwsgiconf.runtime.locking.Lock method*), 36
reload (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
reload() (*in module uwsgiconf.uwsgi_stub*), 24
reload_brutal (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
remote_addr (*uwsgiconf.options.routing.RouteRule.subjects attribute*), 126
remote_logging (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130

remote_user (*uwsgiconf.options.routing.RouteRule.subjects attribute*), 126
replace_placeholders() (*uwsgiconf.config.Section method*), 52
REQ_COUNT_ERR (*uwsgiconf.options.logging.Logging.vars attribute*), 73
REQ_COUNT_ERR_READ (*uwsgiconf.options.logging.Logging.vars attribute*), 73
REQ_COUNT_ERR_WRITE (*uwsgiconf.options.logging.Logging.vars attribute*), 73
REQ_COUNT_VARS_CGI (*uwsgiconf.options.logging.Logging.vars attribute*), 73
REQ_HTTP_HOST (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQ_METHOD (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQ_REFERER (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQ_REMOTE_ADDR (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQ_REMOTE_USER (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQ_SERVER_PROTOCOL (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQ_SIZE_BODY (*uwsgiconf.options.logging.Logging.vars attribute*), 73
REQ_START_CTIME (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQ_START_FORMATTED (*uwsgiconf.options.logging.Logging.vars attribute*), 73
REQ_START_HUMAN (*uwsgiconf.options.logging.Logging.vars attribute*), 73
REQ_START_TS (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQ_START_UNIX_MS (*uwsgiconf.options.logging.Logging.vars attribute*), 73
REQ_START_UNIX_US (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQ_URI (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQ_USER_AGENT (*uwsgiconf.options.logging.Logging.vars attribute*), 72
REQUEST (*uwsgiconf.options.routing.RouteRule.stages attribute*), 125
request (*uwsgiconf.options.routing.RouteRule.vars attribute*), 125
request (*uwsgiconf.runtime.platform._Platform attribute*), 40
request_id() (*in module uwsgiconf.uwsgi_stub*), 24
request_uri (*uwsgiconf.options.routing.RouteRule.subjects attribute*), 126
REQUEST_URI (*uwsgiconf.options.statics.Statics.expiration_criteria attribute*), 134
request_var (*uwsgiconf.options.logging.Logging.vars attribute*), 74
RESP_COUNT_HEADERS (*uwsgiconf.options.logging.Logging.vars attribute*), 74
RESP_SIZE (*uwsgiconf.options.logging.Logging.vars attribute*), 73
RESP_SIZE_BODY (*uwsgiconf.options.logging.Logging.vars attribute*), 73
RESP_SIZE_HEADERS (*uwsgiconf.options.logging.Logging.vars attribute*), 73
RESP_STATUS (*uwsgiconf.options.logging.Logging.vars attribute*), 73
RESP_TIME_MS (*uwsgiconf.options.logging.Logging.vars attribute*), 73
RESP_TIME_US (*uwsgiconf.options.logging.Logging.vars attribute*), 73
RESPONSE (*uwsgiconf.options.routing.RouteRule.stages attribute*), 125
response (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
ResultProcessed (*class in uwsgiconf.runtime.spooler*), 49
ResultRescheduled (*class in uwsgiconf.runtime.spooler*), 49
ResultSkipped (*class in uwsgiconf.runtime.spooler*), 49
rewrite (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127

route() (*in module uwsgiconf.uwsgi_stub*), 24
route_external (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
route_rule (*uwsgiconf.options.routing.Routing attribute*), 128
route_uwsgi (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
RouteAction (*class in uwsgiconf.options.routing_actions*), 104
RouterBase (*class in uwsgiconf.options.routing_routers*), 109
RouterFast (*class in uwsgiconf.options.routing_routers*), 113
RouterForkPty (*class in uwsgiconf.options.routing_routers*), 116
RouterHttp (*class in uwsgiconf.options.routing_routers*), 110
RouterHttps (*class in uwsgiconf.options.routing_routers*), 112
RouterRaw (*class in uwsgiconf.options.routing_routers*), 115
RouterSsl (*class in uwsgiconf.options.routing_routers*), 113
RouterTunTap (*class in uwsgiconf.options.routing_routers*), 117
RouteRule (*class in uwsgiconf.options.routing*), 124
RouteRule.actions (*class in uwsgiconf.options.routing*), 126
RouteRule.stages (*class in uwsgiconf.options.routing*), 125
RouteRule.subjects (*class in uwsgiconf.options.routing*), 125
RouteRule.transforms (*class in uwsgiconf.options.routing*), 126
RouteRule.var_functions (*class in uwsgiconf.options.routing*), 125
RouteRule.vars (*class in uwsgiconf.options.routing*), 124
Routing (*class in uwsgiconf.options.routing*), 128
routing (*uwsgiconf.config.Section attribute*), 51
Routing.modifiers (*class in uwsgiconf.options.routing*), 129
Routing.routers (*class in uwsgiconf.options.routing*), 128
rpc (*uwsgiconf.options.routing.Routing.modifiers attribute*), 130
rpc() (*in module uwsgiconf.uwsgi_stub*), 24
rpc_list() (*in module uwsgiconf.uwsgi_stub*), 24
rrdtool (*uwsgiconf.options.monitoring.Monitoring.pushers attribute*), 93
rsyslog (*uwsgiconf.options.logging.Logging.loggers attribute*), 68
run_command_as_worker() (*uwsgiconf.options.workers.Workers method*), 142
run_command_on_event() (*uwsgiconf.options.main_process.MainProcess method*), 81
run_command_on_touch() (*uwsgiconf.options.main_process.MainProcess method*), 81
run_module() (*uwsgiconf.options.python.Python method*), 148

S

scgi (*uwsgiconf.options.networking.Networking.sockets attribute*), 101
scrolls() (*in module uwsgiconf.uwsgi_stub*), 29
Section (*class in uwsgiconf.config*), 50
Section (*class in uwsgiconf.presets.nice*), 10
Section.embedded_plugins_presets (*class in uwsgiconf.config*), 51
Section.vars (*class in uwsgiconf.config*), 54
send (*uwsgiconf.options.routing.RouteRule.actions attribute*), 127
send() (*in module uwsgiconf.uwsgi_stub*), 24
send() (*uwsgiconf.runtime.mules.Farm method*), 39
send() (*uwsgiconf.runtime.mules.Mule method*), 38
send() (*uwsgiconf.runtime.signals.Signal method*), 46
send_message_raw() (*uwsgiconf.runtime.spooler.Spooler class method*), 48
send_to_spooler() (*in module uwsgiconf.uwsgi_stub*), 25
SENDFILE (*uwsgiconf.options.statics.Statics.transfer_modes attribute*), 134
sendfile() (*in module uwsgiconf.uwsgi_stub*), 25
serve_static (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set() (*uwsgiconf.runtime.caching.Cache method*), 33
set() (*uwsgiconf.runtime.monitoring.Metric method*), 37
set_app_args() (*uwsgiconf.options.python.Python method*), 147
set_autoreload_params() (*uwsgiconf.options.python.Python method*), 148
set_basic_params() (*uwsgiconf.options.alarms.Alarms method*), 56
set_basic_params() (*uwsgiconf.options.applications.Applications method*), 57
set_basic_params() (*uwsgiconf.options.caching.Caching method*), 59
set_basic_params() (*uwsgiconf.options.locks.Locks method*), 64

set_basic_params() (*uwsgiconf.options.logging.Logging method*), 69
set_basic_params() (*uwsgiconf.options.main_process.MainProcess method*), 79
set_basic_params() (*uwsgiconf.options.master_process.MasterProcess method*), 82
set_basic_params() (*uwsgiconf.options.networking.Networking method*), 101
set_basic_params() (*uwsgiconf.options.python.Python method*), 146
set_basic_params() (*uwsgiconf.options.routing_routers.RouterFast method*), 114
set_basic_params() (*uwsgiconf.options.routing_routers.RouterForkPty method*), 116
set_basic_params() (*uwsgiconf.options.routing_routers.RouterHttp method*), 110
set_basic_params() (*uwsgiconf.options.routing_routers.RouterTunTap method*), 117
set_basic_params() (*uwsgiconf.options.spooler.Spooler method*), 132
set_basic_params() (*uwsgiconf.options.statics.Statics method*), 134
set_basic_params() (*uwsgiconf.options.workers.Workers method*), 141
set_basic_params() (*uwsgiconf.options.workers_cheapening.AlgoBusyness method*), 139
set_basic_params() (*uwsgiconf.options.workers_cheapening.AlgoQueue method*), 139
set_basic_params() (*uwsgiconf.options.workers_cheapening.AlgoSpare method*), 139
set_basic_params() (*uwsgiconf.options.workers_cheapening.AlgoSpare2 method*), 139
set_basic_params() (*uwsgiconf.options.workers_cheapening.Cheapening method*), 141
set_bsd_socket_params() (*uwsgiconf.options.networking.Networking method*), 102
set_client_params() (*uwsgiconf.options.subscriptions.Subscriptions method*), 137
set_connections_params() (*uwsgiconf.options.routing_routers.RouterFast method*), 115
set_connections_params() (*uwsgiconf.options.routing_routers.RouterForkPty method*), 116
set_connections_params() (*uwsgiconf.options.routing_routers.RouterHttp method*), 111
set_connections_params() (*uwsgiconf.options.routing_routers.RouterRaw method*), 115
set_connections_params() (*uwsgiconf.options.routing_routers.RouterSsl method*), 113
set_count_auto() (*uwsgiconf.options.workers.Workers method*), 143
set_emergency_params() (*uwsgiconf.options.workers_cheapening.AlgoBusyness method*), 140
set_emperor_command_params() (*uwsgiconf.options.empire.Empire method*), 62
set_emperor_params() (*uwsgiconf.options.empire.Empire method*), 62
set_error_page() (*uwsgiconf.options.routing.Routing method*), 131
set_error_pages() (*uwsgiconf.options.routing.Routing method*), 131
set_exception_handling_params() (*uwsgiconf.options.master_process.MasterProcess method*), 83
set_exit_events() (*uwsgiconf.options.master_process.MasterProcess method*), 82
set_fallback() (*uwsgiconf.config.Section method*), 53
set_file_params() (*uwsgiconf.options.logging.Logging method*), 69
set_filters() (*uwsgiconf.options.logging.Logging method*), 70
set_geoip_params() (*uwsgiconf.options.routing.Routing method*), 132
set_harakiri (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_harakiri_params() (*uwsgiconf.options.workers.Workers method*), 145
set_hook() (*uwsgiconf.options.main_process.MainProcess method*), 80
set_hook_after_request() (*uwsgiconf.options.main_process.MainProcess method*), 80
set_hook_touch() (*uwsgiconf.options.main_process.MainProcess method*), 80
set_host_name (*uwsgi-*

conf.options.main_process.MainProcess.actions attribute), 77
set_idle_params() (*uwsgiconf.options.master_process.MasterProcess method*), 83
set_ipcsem_params() (*uwsgiconf.options.locks.Locks method*), 64
set_logvar() (*in module uwsgiconf.uwsgi_stub*), 25
set_manage_params() (*uwsgiconf.options.routing_routers.RouterHttp method*), 111
set_master_logging_params() (*uwsgiconf.options.logging.Logging method*), 71
set_memory_limits() (*uwsgiconf.options.workers_cheapening.Cheapening method*), 141
set_memory_params() (*uwsgiconf.options.main_process.MainProcess method*), 79
set_metrics_params() (*uwsgiconf.options.monitoring.Monitoring method*), 94
set_metrics_threshold() (*uwsgiconf.options.monitoring.Monitoring method*), 94
set_mode_broodlord_params() (*uwsgiconf.options.empire.Empire method*), 63
set_mode_tyrant_params() (*uwsgiconf.options.empire.Empire method*), 63
set_mules_params() (*uwsgiconf.options.workers.Workers method*), 143
set_naming_params() (*uwsgiconf.options.main_process.MainProcess method*), 81
set_on_exit_params() (*uwsgiconf.options.main_process.MainProcess method*), 80
set_owner_params() (*uwsgiconf.options.main_process.MainProcess method*), 80
set_owner_params() (*uwsgiconf.options.routing_routers.RouterFast method*), 115
set_owner_params() (*uwsgiconf.options.routing_routers.RouterHttp method*), 112
set_paths_caching_params() (*uwsgiconf.options.statics.Statics method*), 135
set_period() (*uwsgiconf.runtime.spooler.Spooler class method*), 48
set_pid_file() (*uwsgiconf.options.main_process.MainProcess method*), 81
set_placeholder() (*uwsgiconf.config.Section method*), 53
set_plugins_params() (*uwsgiconf.config.Section method*), 53
set_postbuffering_params() (*uwsgiconf.options.routing_routers.RouterFast method*), 115
set_reload_on_exception_params() (*uwsgiconf.options.workers.Workers method*), 145
set_reload_params() (*uwsgiconf.options.master_process.MasterProcess method*), 83
set_reload_params() (*uwsgiconf.options.workers.Workers method*), 144
set_requests_filters() (*uwsgiconf.options.logging.Logging method*), 70
set_resubscription_params() (*uwsgiconf.options.routing_routers.RouterFast method*), 114
set_runtime_dir() (*uwsgiconf.config.Section method*), 52
set_script_file (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_server_params() (*uwsgiconf.options.subscriptions.Subscriptions method*), 136
set_server_verification_params() (*uwsgiconf.options.subscriptions.Subscriptions method*), 137
set_sni_dir_params() (*uwsgiconf.options.networking.Networking method*), 104
set_sni_params() (*uwsgiconf.options.networking.Networking method*), 103
set_socket_params() (*uwsgiconf.options.networking.Networking method*), 102
set_spooler_frequency() (*in module uwsgiconf.uwsgi_stub*), 26
set_ssl_params() (*uwsgiconf.options.networking.Networking method*), 103
set_stats_params() (*uwsgiconf.options.monitoring.Monitoring method*), 95
set_thread_params() (*uwsgiconf.options.workers.Workers method*), 143
set_throttle_params() (*uwsgiconf.options.empire.Empire method*), 63
set_tolerance_params() (*uwsgiconf.options.empire.Empire method*), 63
set_unix_socket_params() (*uwsgiconf.options.networking.Networking method*),

102
set_user_harakiri() (*in module uwsgiconf.uwsgi_stub*), 26
set_uwsgi_process_name (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_var_document_root (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_var_path_info (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_var_remote_addr (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_var_remote_user (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_var_request_method (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_var_request_uri (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_var_script_name (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_var_uwsgi_appid (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_var_uwsgi_home (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_var_uwsgi_scheme (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
set_vassals_wrapper_params() (*uwsgiconf.options.empire.Empire method*), 62
set_warning_message() (*in module uwsgiconf.uwsgi_stub*), 26
set_window_params() (*uwsgiconf.options.routing_routers.RouterForkPty method*), 116
set_wsgi_params() (*uwsgiconf.options.python.Python method*), 147
set_zerg_client_params() (*uwsgiconf.options.workers.Workers method*), 146
set_zerg_server_params() (*uwsgiconf.options.workers.Workers method*), 145
setprocname() (*in module uwsgiconf.uwsgi_stub*), 26
shared (*uwsgiconf.options.networking.Networking.sockets attribute*), 101
Signal (*class in uwsgiconf.runtime.signals*), 45
signal (*uwsgiconf.options.alarms.Alarms.alarm_types attribute*), 56
signal (*uwsgiconf.options.routing.RouteRule.actions attribute*), 128
signal (*uwsgiconf.options.routing.Routing.modifiers attribute*), 131
signal() (*in module uwsgiconf.uwsgi_stub*), 26
signal_received() (*in module uwsgiconf.uwsgi_stub*), 26
signal_registered() (*in module uwsgiconf.uwsgi_stub*), 26
signal_wait() (*in module uwsgiconf.uwsgi_stub*), 26
SignalDescription (*class in uwsgiconf.runtime.signals*), 45
SIZE_PACKET_UWSGI (*uwsgiconf.options.logging.Logging.vars attribute*), 74
snmp (*uwsgiconf.options.routing.Routing.modifiers attribute*), 131
Socket (*class in uwsgiconf.options.networking_sockets*), 96
socket (*uwsgiconf.options.logging.Logging.loggers attribute*), 68
socket (*uwsgiconf.options.monitoring.Monitoring.pushers attribute*), 93
SocketDefault (*class in uwsgiconf.options.networking_sockets*), 96
SocketFastcgi (*class in uwsgiconf.options.networking_sockets*), 99
SocketHttp (*class in uwsgiconf.options.networking_sockets*), 96
SocketHttps (*class in uwsgiconf.options.networking_sockets*), 97
SocketRaw (*class in uwsgiconf.options.networking_sockets*), 99
sockets (*in module uwsgiconf.uwsgi_stub*), 15
SocketScgi (*class in uwsgiconf.options.networking_sockets*), 99
SocketShared (*class in uwsgiconf.options.networking_sockets*), 100
SocketUdp (*class in uwsgiconf.options.networking_sockets*), 98
SocketUwsgi (*class in uwsgiconf.options.networking_sockets*), 97
SocketUwsgis (*class in uwsgiconf.options.networking_sockets*), 98
SocketZeromq (*class in uwsgiconf.options.networking_sockets*), 100
spare (*uwsgiconf.options.workers_cheapening.Cheapening.algorithms attribute*), 140
spare2 (*uwsgiconf.options.workers_cheapening.Cheapening.algorithms attribute*), 140
spool() (*in module uwsgiconf.uwsgi_stub*), 26

SPOOL_IGNORE (*in module uwsgiconf.uwsgi_stub*), 14
SPOOL_OK (*in module uwsgiconf.uwsgi_stub*), 14
SPOOL_RETRY (*in module uwsgiconf.uwsgi_stub*), 14
Spooler (*class in uwsgiconf.options.spooler*), 132
Spooler (*class in uwsgiconf.runtime.spooler*), 47
spooler (*in module uwsgiconf.uwsgi_stub*), 15
spooler (*uwsgiconf.config.Section attribute*), 51
spooler (*uwsgiconf.options.routing.Routing.modifiers attribute*), 131
spooler_get_task() (*in module uwsgiconf.uwsgi_stub*), 27
spooler_jobs() (*in module uwsgiconf.uwsgi_stub*), 27
spooler_pid() (*in module uwsgiconf.uwsgi_stub*), 27
spooler_pids() (*in module uwsgiconf.uwsgi_stub*), 27
spooler_task_types (*in module uwsgiconf.runtime.spooler*), 47
SpoolerFunctionCallTask (*class in uwsgiconf.runtime.spooler*), 49
SpoolerTask (*class in uwsgiconf.runtime.spooler*), 49
ssl (*uwsgiconf.options.routing.Routing.routers attribute*), 129
start_response (*in module uwsgiconf.uwsgi_stub*), 15
started_on (*in module uwsgiconf.uwsgi_stub*), 15
started_on (*uwsgiconf.runtime.platform._Platform attribute*), 40
startswith() (*uwsgiconf.options.routing_subjects.SubjectCustom method*), 122
Statics (*class in uwsgiconf.options.statics*), 133
statics (*uwsgiconf.config.Section attribute*), 51
Statics.expiration_criteria (*class in uwsgiconf.options.statics*), 134
Statics.transfer_modes (*class in uwsgiconf.options.statics*), 134
statsd (*uwsgiconf.options.monitoring.Monitoring.pushers attribute*), 93
status (*uwsgiconf.options.routing.RouteRule.subjects attribute*), 126
stdio (*uwsgiconf.options.logging.Logging.loggers attribute*), 68
stop() (*in module uwsgiconf.uwsgi_stub*), 27
SUB_DEFAULT (*uwsgiconf.options.routing_modifiers.ModifierRpc attribute*), 121
SUB_DELETE (*uwsgiconf.options.routing_modifiers.ModifierCache attribute*), 121
SUB_DICT (*uwsgiconf.options.routing_modifiers.ModifierRpc attribute*), 121
SUB_DICT_BASED (*uwsgiconf.options.routing_modifiers.ModifierCache attribute*), 121
SUB_DUMP (*uwsgiconf.options.routing_modifiers.ModifierCache attribute*), 121
SUB_GET (*uwsgiconf.options.routing_modifiers.ModifierCache attribute*), 120
SUB_JSONRPC (*uwsgiconf.options.routing_modifiers.ModifierRpc attribute*), 121
SUB_MAGIC (*uwsgiconf.options.routing_modifiers.ModifierCache attribute*), 121
SUB_PING (*uwsgiconf.options.routing_modifiers.ModifierPing attribute*), 120
SUB_PONG (*uwsgiconf.options.routing_modifiers.ModifierPing attribute*), 120
SUB_RAW (*uwsgiconf.options.routing_modifiers.ModifierRpc attribute*), 121
SUB_RING (*uwsgiconf.options.routing_modifiers.ModifierJvm attribute*), 119
SUB_SET (*uwsgiconf.options.routing_modifiers.ModifierCache attribute*), 120
SUB_STREAM (*uwsgiconf.options.routing_modifiers.ModifierCache attribute*), 121
SUB_USE_PATH_INFO (*uwsgiconf.options.routing_modifiers.ModifierRpc attribute*), 121
SUB_XMLRPC (*uwsgiconf.options.routing_modifiers.ModifierRpc attribute*), 121
SubjectCustom (*class in uwsgiconf.options.routing_subjects*), 122
SubjectHttpHost (*class in uwsgiconf.options.routing_subjects*), 123
SubjectHttpReferer (*class in uwsgiconf.options.routing_subjects*), 123
SubjectHttpUserAgent (*class in uwsgiconf.options.routing_subjects*), 123
SubjectPathInfo (*class in uwsgiconf.options.routing_subjects*), 122
SubjectQueryString (*class in uwsgiconf.options.routing_subjects*), 123
SubjectRemoteAddr (*class in uwsgiconf.options.routing_subjects*), 123
SubjectRemoteUser (*class in uwsgiconf.options.routing_subjects*), 123
SubjectRequestUri (*class in uwsgiconf.options.routing_subjects*), 123
SubjectStatus (*class in uwsgiconf.options.routing_subjects*), 123
subscribe() (*uwsgiconf.options.subscriptions.Subscriptions method*), 138
subscription (*uwsgiconf.options.routing.Routing.modifiers attribute*), 131
Subscriptions (*class in uwsgi-*

conf.options.subscriptions), 136
subscriptions (*uwsgiconf.config.Section attribute*), 51
Subscriptions.algorithms (*class in uwsgiconf.options.subscriptions*), 136
suffix (*uwsgiconf.options.logging.Logging.encoders attribute*), 69
sum (*uwsgiconf.options.monitoring.Monitoring.collectors attribute*), 93
suspend() (*in module uwsgiconf.uwsgi_stub*), 27
switch_into_lazy_mode() (*uwsgiconf.options.applications.Applications method*), 58
SymbolsImporter (*in module uwsgiconf.uwsgi_stub*), 14
SymbolsZipImporter (*in module uwsgiconf.uwsgi_stub*), 14
symcall (*uwsgiconf.options.routing.Routing.modifiers attribute*), 131
syslog (*uwsgiconf.options.logging.Logging.loggers attribute*), 68

T

target (*uwsgiconf.runtime.signals.SignalDescription attribute*), 45
task() (*uwsgiconf.runtime.spooler.Spooler method*), 48
TaskResult (*class in uwsgiconf.runtime.spooler*), 49
template (*uwsgiconf.options.routing.RouteRule.transforms attribute*), 126
threads_enabled (*uwsgiconf.runtime.platform._Platform attribute*), 40
TIME (*uwsgiconf.options.logging_encoders.EncoderFormat.vars attribute*), 67
time (*uwsgiconf.options.routing.RouteRule.vars attribute*), 125
TIME_FORMAT (*uwsgiconf.options.logging_encoders.EncoderFormat.vars attribute*), 67
TIME_MS (*uwsgiconf.options.logging_encoders.EncoderFormat.vars attribute*), 67
TIME_UNIX (*uwsgiconf.options.logging.Logging.vars attribute*), 74
TIME_US (*uwsgiconf.options.logging_encoders.EncoderFormat.vars attribute*), 67
TimeFormatter (*class in uwsgiconf.options.logging_encoders*), 67
TIMESTAMP_STARTUP_MS (*uwsgiconf.config.Section.vars attribute*), 54
TIMESTAMP_STARTUP_S (*uwsgiconf.config.Section.vars attribute*), 54
to_file (*uwsgiconf.options.routing.RouteRule.transforms attribute*), 126
tofile() (*uwsgiconf.config.Configuration method*), 55
total_count (*uwsgiconf.runtime.request._Request attribute*), 41
total_requests() (*in module uwsgiconf.uwsgi_stub*), 27
tuntap (*uwsgiconf.options.routing.Routing.routers attribute*), 129

U

udp (*uwsgiconf.options.networking.Networking.sockets attribute*), 101
unbit (*in module uwsgiconf.uwsgi_stub*), 15
unlink (*uwsgiconf.options.main_process.MainProcess.actions attribute*), 77
unlock() (*in module uwsgiconf.uwsgi_stub*), 27
upper (*uwsgiconf.options.routing.RouteRule.transforms attribute*), 126
upper (*uwsgiconf.options.routing.RouteRule.var_functions attribute*), 125
use_router() (*uwsgiconf.options.routing.Routing method*), 131
USER_ID (*uwsgiconf.config.Section.vars attribute*), 54
USER_NAME (*uwsgiconf.config.Section.vars attribute*), 54
uwsgi (*uwsgiconf.options.networking.Networking.sockets attribute*), 101
uwsgi (*uwsgiconf.options.routing.RouteRule.vars attribute*), 125
uwsgiconf.config (*module*), 50
uwsgiconf.formatters (*module*), 148
uwsgiconf.options.alarm_types (*module*), 56
uwsgiconf.options.alarms (*module*), 56
uwsgiconf.options.applications (*module*), 57
uwsgiconf.options.caching (*module*), 59
uwsgiconf.options.empire (*module*), 61
uwsgiconf.options.locks (*module*), 64
uwsgiconf.options.logging (*module*), 68
uwsgiconf.options.logging_encoders (*module*), 67
uwsgiconf.options.logging_loggers (*module*), 65
uwsgiconf.options.main_process (*module*), 76
uwsgiconf.options.main_process_actions (*module*), 74
uwsgiconf.options.master_process (*module*), 82
uwsgiconf.options.monitoring (*module*), 92
uwsgiconf.options.monitoring_collectors (*module*), 91
uwsgiconf.options.monitoring_metric_types (*module*), 86
uwsgiconf.options.monitoring_pushers (*module*), 89

uwsgiconf.options.networking (*module*), 100
uwsgiconf.options.networking_sockets (*module*), 96
uwsgiconf.options.python (*module*), 146
uwsgiconf.options.queue (*module*), 104
uwsgiconf.options.routing (*module*), 124
uwsgiconf.options.routing_actions (*module*), 104
uwsgiconf.options.routing_modifiers (*module*), 118
uwsgiconf.options.routing_routers (*module*), 109
uwsgiconf.options.routing_subjects (*module*), 122
uwsgiconf.options.routing_vars (*module*), 123
uwsgiconf.options.spooler (*module*), 132
uwsgiconf.options.statics (*module*), 133
uwsgiconf.options.subscriptions (*module*), 136
uwsgiconf.options.subscriptions_algos (*module*), 136
uwsgiconf.options.workers (*module*), 141
uwsgiconf.options.workers_cheapening (*module*), 139
uwsgiconf.presets.empire (*module*), 9
uwsgiconf.presets.nice (*module*), 10
uwsgiconf.runtime.alarms (*module*), 32
uwsgiconf.runtime.asynced (*module*), 32
uwsgiconf.runtime.caching (*module*), 33
uwsgiconf.runtime.control (*module*), 34
uwsgiconf.runtime.locking (*module*), 35
uwsgiconf.runtime.logging (*module*), 36
uwsgiconf.runtime.monitoring (*module*), 36
uwsgiconf.runtime.mules (*module*), 38
uwsgiconf.runtime.rpc (*module*), 41
uwsgiconf.runtime.scheduling (*module*), 42
uwsgiconf.runtime.signals (*module*), 45
uwsgiconf.runtime.spooler (*module*), 47
uwsgiconf.uwsgi_stub (*module*), 14
uwsgis (*uwsgiconf.options.networking.Networking.sockets attribute*), 101

V

v8 (*uwsgiconf.options.routing.Routing.modifiers attribute*), 131
value (*uwsgiconf.runtime.monitoring.Metric attribute*), 37
Var (*class in uwsgiconf.options.logging*), 68
Var (*class in uwsgiconf.options.routing_vars*), 123
VarCookie (*class in uwsgiconf.options.routing_vars*), 123
VarGeoip (*class in uwsgiconf.options.routing_vars*), 123
VarHttptime (*class in uwsgiconf.options.routing_vars*), 124
variable_get() (*in module uwsgiconf.runtime.logging*), 36
VarMetric (*class in uwsgiconf.options.logging*), 68
VarMetric (*class in uwsgiconf.options.routing_vars*), 123
VarQuery (*class in uwsgiconf.options.routing_vars*), 123
VarRequest (*class in uwsgiconf.options.routing_vars*), 123
VarRequestVar (*class in uwsgiconf.options.logging*), 68
vars_city (*uwsgiconf.options.routing_vars.VarGeoip attribute*), 123
vars_country (*uwsgiconf.options.routing_vars.VarGeoip attribute*), 123
VarTime (*class in uwsgiconf.options.routing_vars*), 124
VarUwsgi (*class in uwsgiconf.options.routing_vars*), 123
VASSAL_CONFIG_CHANGE_POST (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
VASSAL_ON_DEMAND_IN (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
VASSAL_PRIV_DRP_PRE (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
VASSAL_SET_NAMESPACE (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
VASSAL_START_IN (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
VASSAL_START_POST (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
VASSAL_START_PRE (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
version (*in module uwsgiconf.uwsgi_stub*), 15
VERSION (*uwsgiconf.config.Section.vars attribute*), 54
version_info (*in module uwsgiconf.uwsgi_stub*), 15

W

wait() (*uwsgiconf.runtime.signals.Signal method*), 47
wait_fd_read() (*in module uwsgiconf.uwsgi_stub*), 28
wait_fd_write() (*in module uwsgiconf.uwsgi_stub*), 28
websocket_handshake() (*in module uwsgiconf.uwsgi_stub*), 28

websocket_recv() (*in module uwsgiconf.uwsgi_stub*), 28
websocket_recv_nb() (*in module uwsgiconf.uwsgi_stub*), 28
websocket_send() (*in module uwsgiconf.uwsgi_stub*), 28
websocket_send_binary() (*in module uwsgiconf.uwsgi_stub*), 29
weighted_least_reference_count (*uwsgiconf.options.subscriptions.Subscriptions.algorithms attribute*), 136
weighted_round_robin (*uwsgiconf.options.subscriptions.Subscriptions.algorithms attribute*), 136
WeightedLeastReferenceCount (*class in uwsgiconf.options.subscriptions_algos*), 136
WeightedRoundRobin (*class in uwsgiconf.options.subscriptions_algos*), 136
WORKER_ACCEPTING_PRE_EACH (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
WORKER_ACCEPTING_PRE_EACH_ONCE (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
WORKER_ACCEPTING_PRE_FIRST (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 78
WORKER_ACCEPTING_PRE_FIRST_ONCE (*uwsgiconf.options.main_process.MainProcess.phases attribute*), 79
WORKER_ID (*uwsgiconf.options.logging.Logging.vars attribute*), 74
worker_id (*uwsgiconf.runtime.platform._Platform attribute*), 40
worker_id() (*in module uwsgiconf.uwsgi_stub*), 29
WORKER_PID (*uwsgiconf.options.logging.Logging.vars attribute*), 74
Workers (*class in uwsgiconf.options.workers*), 141
workers (*uwsgiconf.config.Section attribute*), 51
workers() (*in module uwsgiconf.uwsgi_stub*), 29
workers_count (*uwsgiconf.runtime.platform._Platform attribute*), 40
workers_info (*uwsgiconf.runtime.platform._Platform attribute*), 40
wsgi (*uwsgiconf.options.routing.Routing.modifiers attribute*), 131

X

xmpp (*uwsgiconf.options.alarms.Alarms.alarm_types attribute*), 56
xslt (*uwsgiconf.options.routing.Routing.modifiers attribute*), 131

Z

zabbix (*uwsgiconf.options.monitoring.Monitoring.pushers attribute*), 93
zeromq (*uwsgiconf.options.logging.Logging.loggers attribute*), 68
zeromq (*uwsgiconf.options.networking.Networking.sockets attribute*), 101
ZipImporter (*in module uwsgiconf.uwsgi_stub*), 15