logconfig Documentation

Release stable

January 26, 2015

Contents

Simple helper moudle for configuring Python logging.

CHAPTER 1

Requirements

1.1 Compatibility

- Python 2.6
- Python 2.7
- Python 3.2
- Python 3.3
- Python 3.4

1.2 Dependencies

- PyYAML
- logutils (if using Python 2)

CHAPTER 2

Installation

```
pip install logconfig
```

CHAPTER 3

Overview

This simple library exposes several helper methods for configuring the standard library's `logging` module. There's nothing fancy about it. Under the hood `logconfig` uses `logging.config` to load various configuartion formats.

In addition to configuration loading, `logconfig` provides helpers for easily converting a configured logger's handlers utilize a queue.

3.1 Supported Configuration Formats

- JSON
- YAML
- ConfigParser
- Python Dict

CHAPTER 4

Quickstart

4.1 Configuration Loading

```
import logconfig
import logging

# Load config from JSON file
logconfig.from_json('path/to/file.json')

# Load config from YAML file
logconfig.from_yaml('path/to/file.yml')

# Load config from ConfigParser file
logconfig.from_yaml('path/to/file.cfg')

# Load config from dict
logconfig.from_dict(config_dict)

log = logging.getLogger()
log.debug('Configuration loaded using logconfig')
```

4.2 Queue Utilization

```
import logconfig
import logging

logconfig.from_dict({
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'level': 'DEBUG'
        }
    },
    'loggers': {
        'mylogger': {
            'handlers': ['console']
        }
    }
```

```
})

# Convert logger's handlers to utilize a queue
queue = logconfig.Queue(-1)
listener = logconfig.QueueListener(queue)
handler = logconfig.QueueHandler(queue)

mylogger = logging.getLogger('mylogger')

# You can also pass in the logger name instead of the actual logger.
# logconfig.queuify_logger('mylogger', handler, listener)
logconfig.queuify_logger(mylogger, handler, listener)

assert isinstance(mylogger.handlers[0], logconfig.QueueHandler)

# Start the listener.
listener.start()

# When finished, stop the listener.
# This is optional, but not doing so may prevent some logs from being processed.
listener.stop()
```

CHAPTER 5

Usage

Use `logconfig` to easily load `logging` configurations. For more details on configuring `logging`, visit https://docs.python.org/library/logging.config.html.

```
import logconfig
```

5.1 Configuration from JSON

Configure logging using JSON file.

```
logconfig.from_json(filename)
```

Example JSON file:

```
{
    "version": 1,
    "disable_existing_loggers": false,
    "formatters": {
        "simple": {
            "format": "%(asctime)s. - %(name)s - %(levelname)s - %(message)s"
        }
    },
    "handlers": {
        "console": {
            "class": "logging.StreamHandler",
            "level": "DEBUG",
            "formatter": "simple",
            "stream": "ext://sys.stdout"
        }
    },
    "root": {
        "level": "DEBUG",
        "handlers": ["console"]
    }
}
```

5.2 Configuration from YAML

Configure logging using YAML file.

```
logconfig.from_yaml(filename)
```

Example YAML file:

```
version: 1
disable_existing_loggers: False
formatters:
  simple:
    format: "%(asctime)s. - %(name)s - %(levelname)s - %(message)s"
handlers:
  console:
    class: logging.StreamHandler
    level: DEBUG
    formatter: simple
    stream: ext://sys.stdout
root:
  level: DEBUG
  handlers: [console]
```

5.3 Configuration from ConfigParser File

Configure logging using ConfigParser compatible file.

```
logconfig.from_file(filename)
```

Example CFG file:

```
[loggers]
keys=root

[handlers]
keys=console

[formatters]
keys=simple

[logger_root]
level=DEBUG
handlers=console

[handler_console]
class=StreamHandler
level=DEBUG
formatter=simple
args=(sys.stdout,)

[formatter_simple]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
```

5.4 Configuration from Dict

Configure logging using Python dictionary.

```
logconfig.from_dict(dct)
```

Example dict:

```
{
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'simple': {
            'format': '%(asctime)s. - %(name)s - %(levelname)s - %(message)s'
        }
    },
    'handlers': {
        'console': {
            'formatter': 'simple',
            'class': 'logging.StreamHandler',
            'level': 'DEBUG',
            'stream': 'ext://sys.stdout'
        }
    },
    'root': {
        'handlers': ['console'],
        'level': 'DEBUG'
    }
}
```

5.5 Configuration from Autodetection

If, for whatever reason, you do not know what the source of the configuration will be (or if you're just feeling lucky), then you can try to coerce logging configuration using one of the autodetection methods:

```
logconfig.from_filename(filename)
logconfig.from_autodetect(filename_or_dict)
```

```
try:
    logconfig.from_filename(filename)
    logconfig.from_autodetect(filename_or_dict)
except logconfig.LogConfigException as ex:
    # Unrecognized configuration argument.
    pass
```

These methods will try to dispatch the function argument to the proper configuration loader or fail trying.

5.6 Configuration from Environment Variable

Configure logging using filename provided via environment variable.

```
logconfig.from_env(variable_name)
```

NOTE: Environment variable value will be passed to `from_filename()`.