

---

# ProsperCommon Documentation

*Release 1.0.2*

**John Purcell**

**Jul 22, 2020**



---

## Contents:

---

<b>1</b>	<b>Quickstart</b>	<b>3</b>
<b>2</b>	<b>Features</b>	<b>5</b>
<b>3</b>	<b>Index</b>	<b>7</b>
3.1	Getting Started . . . . .	7
3.2	ProsperLogger . . . . .	8
3.3	ProsperConfig . . . . .	11
3.4	ProsperVersion . . . . .	11
3.5	ProsperCLI . . . . .	12
<b>4</b>	<b>API Reference</b>	<b>15</b>
4.1	common package . . . . .	15
<b>5</b>	<b>Indices and tables</b>	<b>27</b>
<b>Python Module Index</b>		<b>29</b>
<b>Index</b>		<b>31</b>



Common utilities for EVEProsper project. To reduce boilerplate across the shared projects in the [EVEProsper](#) toolset



# CHAPTER 1

---

## Quickstart

---

Install ProsperCommon

```
pip install ProsperCommon
```



# CHAPTER 2

---

## Features

---

- `prosper_logging`: logging helpers/builders for Prosper projects
- `prosper_config`: unified configparser to pick the right values and keep secrets off github
- `prosper_version`: helpers to resolve correct package version
- `prosper_cli`: framework for creating CLI applications



# CHAPTER 3

---

## Index

---

## 3.1 Getting Started

ProsperCommon is a group of utilities we expect nearly every Prosper project to use. These libraries are designed to be easy and uniform.

### 3.1.1 Using ProsperCommon

```
pip install ProsperCommon
```

It's worth noting that nearly every Prosper project requires a `app_config.cfg` file. Some libraries will expect certain namespaces be reserved.

EX: `prosper.common.prosper_logging.ProsperLogger()` expects to use `[LOGGING]` section for config keeping

Please review documentation carefully to avoid namespace collisions

### 3.1.2 Updating ProsperCommon

If working from dev/GitHub:

```
pip install -e .
```

### Testing

```
python setup.py test
```

Since common is so important to Prosper projects, testing and coverage are extremely important. Please know PR's will require the following

- >90% coverage

- PEP8 compliance
- Napoleon style docstrings for all functions and classes
- Docs coverage for any new functionality

## Docs

```
pip install .[dev]
sphinx-apidoc -f -o docs/source prosper/common/ Update autodocs
sphinx-build -b html docs/ webpage/ build docs
```

Documentation is important. Please make sure to update docs before release

## Release

Release is handled by tagging + [Travis-CI](#). Tagged versions are automatically pushed to PyPI.

Release message should include useful update notes, and versions should follow [Semantic Versioning](#) standard.

## 3.2 ProsperLogger

All Prosper scripts use a unified logger. ProsperLogger is the easy way to build/extend any logger. Death to `print()` long live logger

### 3.2.1 ProsperLogger()

Building a logger is easy.

```
import prosper.common.prosper_logging as p_logging

LogBuilder = p_logging.ProsperLogger(
    'log_name',
    'desired/log/path',
    configuration_object,
    bool:debug_mode [optional]
)
LogBuilder.configure_discord_logger() # log ERROR/CRITICAL to Discord

if DEBUG:
    LogBuilder.configure_debug_logger() # log debug messages to STDOUT

logger = LogBuilder.get_logger()
```

the LogBuilder can be extended with some other handlers if required. Also, defaults can be rerun if desired.

### 3.2.2 Built-In Handlers

### `configure_default_logger()`

```

1 def configure_default_logger(
2     log_freq:str,
3     log_total:int,
4     log_level:log_level_str,
5     log_format:log_format_str,
6     debug_mode:bool
7 ) :

```

- `log_freq`: TimedRotatingFileHandler definition
- `log_total`: how many log-periods to retain
- `log_level`: Desired minimum log level
- `log_format`: Python log formatter string
- `debug_mode`: unused at this time

This handler is loaded by default. It can be reset by calling `ProsperLogger().configure_default_logger(...)` again. **THIS SHOULD BE DONE AS EARLY AS POSSIBLE** can wipe out all other attached handlers.

### `configure_debug_logger()`

```

1 def configure_debug_logger(
2     log_level:log_level_str,
3     log_format:log_format_str,
4     debug_mode:bool
5 ) :

```

- `log_level`: default = ‘DEBUG’ (print everything)
- `log_format`: default = `ReportingFormats.STDOUT`
- `debug_mode`: unused

For live debugging, report logging messages to standard out. This can be attached by a `Plumbum.cli` for easy toggling between debug/production logging

### `configure_discord_logger()`

```

1 def configure_discord_logger(
2     discord_webhook:url_str,
3     discord_recipient:'<@int>'_discord_id_str,
4     log_level:log_level_str,
5     log_format:log_format_str,
6     debug_mode:bool
7 ) :

```

- `discord_webhook`: discord webhook url
- `discord_recipients`: <@int> for alerting users/groups (see app developer console)
- `log_level`: default ‘ERROR’
- `log_format`: default `ReportingFormats.PRETTY_PRINT`

- **debug\_mode:** unused

Live alerting is a useful tool. ProsperCommon is loaded with a REST handler for pushing logging alerts to discord webhooks. Any alerts above a given level will be pushed out to a discord channel along the webhook pipeline

#### **configure\_slack\_logger()**

```
1 def configure_slack_logger(
2     slack_webhook:url_str,
3     log_level:log_level_str,
4     log_format:log_format_str
5     debug_mode:bool
6 ) :
```

- **slack\_webhook:** Slack webhook url
- **log\_level:** default ‘ERROR’
- **log\_format:** default ReportingFormats.PRETTY\_PRINT
- **debug\_mode:** unused

Similar to the Discord handler, the Slack handler works very similarly. Just get a [webhook for slack](#) and assign the appropriate channel scope.

**NOTE:** does not have alerting built in by default. Best-practice for alerting humans may be to configure multiple slack\_logger handles with direct message webhooks.

### **3.2.3 Logging Configuration**

ProsperLogger is designed with the following priority order for finding configurations:

1. arguments in `configure_handler` calls
2. `__init__` called `configuration_object` loaded by the script that needs the logger
3. `prosper.common/common_config.cfg` as global defaults

`## configuration_object`

```
[LOGGING]
log_level = INFO
log_path = .
log_freq = midnight
log_total = 30
discord_webhook = #SECRET
discord_level = ERROR
discord_alert_recipient = <@236681427817725954>
slack_webhook = #SECRET
```

This section is valid in any loaded configuration object loaded by `prosper.common.prosper_config.ProsperConfig()`. Any commented/blank keys are loaded as None but should have error handling in place.

### **3.2.4 ReportingFormats**

[Python Log Formats](#) are obnoxious to write, and leaving them in config-levels could lead to version upgrading issues later.

Instead we include some helpful baked-in formats for easy setup:

- `ReportingFormats.DEFAULT` (for file logging)

```
[2016-10-14 16:11:38,805;DEBUG;prosper_logging.py;<module>;185] my debug message
```

- `ReportingFormats.PRETTY_PRINT` (for Discord logging)

```
[DEBUG:prosper_logging.py--<module>;185]
my debug message
```

- `ReportingFormats.STDOUT` (for STDOUT/console logging)

```
[DEBUG:prosper_logging.py--<module>;185] my debug message
```

## 3.3 ProsperConfig

Parsing global/local configs can be obnoxious. We provide a way to use/override configs. Especially for libraries, a way to control globals or override them.

All Prosper use the `ProsperConfig` parser. Powered by py3's `configparser`.

### 3.3.1 ProsperConfig()

```
import prosper.common.prosper_config as p_config

ConfigObj = ProsperConfig(
    'path/to/config.cfg'
    local_filepath_override='path/to/custom_config.cfg' #optional
)

option_value = ConfigObj.get_option('SECTION_NAME', 'KEY_NAME', override_value,
→default_value)
```

This should give the following priority for `option_value`

1. if `override_value != default_value`: `override value`: A value given at arg time
2. `local_config['SECTION_NAME']['KEY_NAME']`: Untracked local config file (secrets safe)
3. `global_config['SECTION_NAME']['KEY_NAME']`: Git tracked config file
4. `os.environ.get('PROSPER_{SECTION_NAME}__{KEY_NAME}')`: Check the environment for values (secrets safe-ish)
5. `default_value` as a final result to avoid returning `None` where it wouldn't be supported

## 3.4 ProsperVersion

Getting version information reliably vs [GitHub](#), [PyPI](#), [Travis-CI](#), and [source](#) is difficult. Taking a page out of `ccgames/setuphelpers` to standardize work.

### 3.4.1 Using prosper\_version

```
_version.py
"""_version.py: a place to report package version info"""
from os import path
import warnings

INSTALLED = True
try:
    import prosper.common.prosper_version as p_version
    #requires helper, but setup.py install can't reference itself on first pass
except ImportError:
    INSTALLED = False

HERE = path.abspath(path.dirname(__file__))

def get_version():
    """find current version

    Returns:
        (str): current version

    """
    if not INSTALLED:
        warnings.warn('Unable to resolve package version until installed', UserWarning)
        return '0.0.0'

    return p_version.get_version(HERE)

__version__ = get_version() #required for setup.py to find via `importlib`
```

This code helps both users of Prosper projects and devs get the version information reliably, without worrying about all the background conflicts.

### 3.4.2 Version Priorities

Below is the rank/order prosper\_version will try to resolve version information

1. TRAVIS\_TAG: if test is running on [Travis-CI](#), default to its version (release)
2. git tag: Look for latest version in git (dev)
3. version.txt: Look for latest version in version.txt file. (PyPI released)
4. default\_version: If all else fails, default to 0.0.0

## 3.5 ProsperCLI

Help create uniform templates and reduce boilerplate, ProsperCLI gives a common jumping off point for any scraper/utility. Based off [Plumbum](#) framework.

### 3.5.1 Using prosper\_cli

#### Generic CLI

```
"""my_app.py"""
from os import path
import prosper.common.prosper_cli as p_cli
from _version import __version__

class MyApplication(p_cli.ProsperApplication):
    PROGNAME = 'my_app_name' # REQUIRED
    VERSION = __version__

    config_path = path.join(
        path.abspath(path.dirname(__file__)),
        'my_config_file.cfg'
    )

    def main(self):
        """actual logic goes here"""
        self.logger.info('Hello world!')
        ...

if __name__ == '__main__':
    MyApplication.run()
```

---

## Flask Launcher

```
"""my_app.py"""
from os import path
import prosper.common.prosper_cli as p_cli
from _version import __version__
from endpoints import APP

class MyFlaskApplication(p_cli.FlaskLauncher):
    PROGNAME = 'my_app_name' # REQUIRED
    VERSION = __version__

    config_path = path.join(
        path.abspath(path.dirname(__file__)),
        'my_config_file.cfg'
    )

    def main(self):
        """actual logic goes here"""
        self.notify_launch()

        APP.run(
            host=self.get_host(),
            port=self.port,
            debug=self.debug,
            threaded=self.threaded,
            process=self.workers,
        )

if __name__ == '__main__':
    MyFlaskApplication.run()
```

Meant to be used with ProsperCookiecutters for debug launching Flask apps.

## CLI Features

By using the Prosper framework, the following is handled automatically:

- Help/version info handled by [Plumbum](#)
- `-d/--debug` bool for avoiding production mode
- `-v/--verbose` bool for enabling STDOUT logging
- `--config` for loading a custom config file
- `--dump-config` for dumping default config to STDOUT
- `--secret-cfg` for using a jinja2 template secret-keeping style
- `self.logger` and `self.config` loaded automagically
- **Full ProsperLogging support**
  - Slack and Discord support if webhooks are provided by config
  - Standardized log formatting
  - Platform and version information for webhook loggers

## Secret Config

Sometimes having a secrets file is preferable to using environment variables. This allows secrets to be more easily passed as keys.

```
# credentials.ini [key]
value_1 = secret value_2 = secret
```

# CHAPTER 4

---

## API Reference

---

### 4.1 common package

#### 4.1.1 Submodules

#### 4.1.2 common.exceptions module

exceptions.py: custom exceptions and warnings for prosper.common libraries

**exception** common.exceptions.**ProsperCommonException**  
Bases: Exception

base class for prosper.common exceptions

**exception** common.exceptions.**ProsperCommonWarning**  
Bases: UserWarning

base class for prosper.common warnings

**exception** common.exceptions.**ProsperDefaultVersionWarning**  
Bases: *common.exceptions.ProsperVersionWarning*

unable to set any version other than default. New project or broken git?

**exception** common.exceptions.**ProsperLoggerWarning**  
Bases: *common.exceptions.ProsperCommonWarning*

base class for prosper.common.prosper\_logging warnings

**exception** common.exceptions.**ProsperVersionException**  
Bases: *common.exceptions.ProsperCommonException*

base class for prosper.common.version exceptions

**exception** common.exceptions.**ProsperVersionTestModeWarning**  
Bases: *common.exceptions.ProsperVersionWarning*

for overriding Travis modes for unit testing coverage

```
exception common.exceptions.ProsperVersionWarning
Bases: common.exceptions.ProsperCommonWarning

base class for prosper.common.prosper_version warnings

exception common.exceptions.WebhookCreateFailed
Bases: common.exceptions.ProsperLoggerWarning

unable to generate webhook requested

exception common.exceptions.WebhookFailedEmitWarning
Bases: common.exceptions.ProsperLoggerWarning

Something went wrong in webhook handler. Warn rather than raise
```

#### 4.1.3 common.prosper\_cli module

Plumbum CLI wrapper for easier/common application writing

```
class common.prosper_cli.FlaskLauncher(executable)
Bases: common.prosper_cli.ProsperApplication

wrapper for launching (DEBUG) Flask apps

get_host()
    returns appropriate host configuration

    Returns host IP (127.0.0.1 or 0.0.0.0)

    Return type str

notify_launch(log_level='ERROR')
    logs launcher message before startup

    Parameters log_level (str) – level to notify at

port
    Sets an attribute

threaded
    Sets an attribute

workers
    Sets an attribute

class common.prosper_cli.ProsperApplication(executable)
Bases: plumbum.cli.application.Application

parent-wrapper for CLI applications

config
    uses “global config” for cfg

debug
    Sets an attribute

dump_config()
    dumps configfile to stdout so users can edit/implement their own

load_secrets(secret_path)
    render secrets into config object

logger
    uses “global logger” for logging
```

```
override_config(config_path)
    override config object with local version

verbose
    Sets an attribute

class common.prosper_cli.ProsperTESTApplication(executable)
    Bases: common.prosper_cli.ProsperApplication

    test wrapper for CLI tests

    HERE = '/home/docs/checkouts/readthedocs.org/user_builds/prospercommon/checkouts/stable'
    PROGNAME = 'CLITEST'
    VERSION = '0.0.0'

    config_path = '/home/docs/checkouts/readthedocs.org/user_builds/prospercommon/checkouts/stable'
    main()
        do stuff
```

#### 4.1.4 common.prosper\_config module

prosper\_config.py

Unified config parsing and option picking against config objects

```
class common.prosper_config.ProsperConfig(config_filename, local_filepath_override="")
    Bases: object
```

configuration handler for all prosper projects

Helps maintain global, local, and args values to pick according to priority

1. args given at runtile
2. <config\_file>.local.cfg – untracked config with #SECRETS
3. <config\_file>.cfg – tracked ‘master’ config without #SECRETS
4. environment varabile
5. args\_default – function default w/o global config

##### Parameters

- **config\_filename** (str) – path to config
- **local\_filepath\_override** (str, optional) – path to alternate private config file

**global\_config**

**Type** configparser.ConfigParser

**local\_config**

**Type** configparser.ConfigParser

**config\_filename**

filename of global/tracked/default .cfg file

**Type** str

**local\_config\_filename**  
filename for local/custom .cfg file

**Type** str

**get** (*section\_name*, *key\_name*)  
Replicate configparser.get() functionality

**Parameters**

- **section\_name** (str) – section name in config
- **key\_name** (str) – key name in config.section\_name

**Returns** do not check defaults, only return local value

**Return type** str

**Raises** KeyError – unable to find option in either local or global config

**get\_option** (*section\_name*, *key\_name*, *args\_option=None*, *args\_default=None*)  
evaluates the requested option and returns the correct value

## Notes

Priority order 1. args given at runtile 2. <config\_file>.local.cfg – untracked config with #SECRETS 3. <config\_file>.cfg – tracked ‘master’ config without #SECRETS 4. environment varabile 5. args\_default – function default w/o global config

**Parameters**

- **section\_name** (str) – section level name in config
- **key\_name** (str) – key name for option in config
- **args\_option** (any) – arg option given by a function
- **args\_default** (any) – arg default given by a function

**Returns** appropriate response as per priority order

**Return type** str

**logger = <Logger ProsperCommon (WARNING)>**  
common.prosper\_config.**check\_value** (*config*, *section*, *option*, *jinja\_pattern=re.compile('.\*{\{\}\}\}.\*)'*)  
try to figure out if value is valid or jinja2 template value

**Parameters**

- **config** (configparser.ConfigParser) – config object to read key from
- **section** (str) – name of section in configparser
- **option** (str) – name of option in configparser
- **jinja\_pattern** (\_sre.SRE\_Pattern) – a *re.compile()* pattern to match on

**Returns** value if value, else None

**Return type** str

**Raises**

- KeyError
- configparser.NoOptionError

- configparser.NoSectionError

common.prosper\_config.**get\_configs**(config\_filepath, local\_filepath\_override="")  
go and fetch the global/local configs from file and load them with configparser

**Parameters**

- **config\_filepath**(str) – path to config
- **local\_filepath\_override**(str) – secondary place to locate config file

**Returns** global\_config ConfigParser: local\_config

**Return type** ConfigParser

common.prosper\_config.**get\_local\_config\_filepath**(config\_filepath, force\_local=False)  
helper for finding local filepath for config

**Parameters**

- **config\_filepath**(str) – path to local config abspath > relpath
- **force\_local**(bool) – force return of \_local.cfg version

**Returns** Path to local config, or global if path DNE

**Return type** str

common.prosper\_config.**get\_value\_from\_environment**(section\_name, key\_name, envname\_pad='PROSPER', logger=<Logger ProsperCommon (WARNING)>)

check environment for key/value pair

**Parameters**

- **section\_name**(str) – section name
- **key\_name**(str) – key to look up
- **envname\_pad**(str) – namespace padding
- **logger**(logging.Logger) – logging handle

**Returns** value in environment

**Return type** str

common.prosper\_config.**read\_config**(config\_filepath, logger=<Logger ProsperCommon (WARNING)>)

fetch and parse config file

**Parameters**

- **config\_filepath**(str) – path to config file. abspath > relpath
- **logger**(logging.Logger) – logger to catch error msgs

common.prosper\_config.**render\_secrets**(config\_path, secret\_path)

combine a jinja template with a secret .ini file

**Parameters**

- **config\_path**(str) – path to .cfg file with jinja templating
- **secret\_path**(str) – path to .ini-like secrets file

**Returns** rendered configuration object

**Return type** *ProsperConfig*

#### 4.1.5 common.prosper\_logging module

prosper\_logging.py

A unified logger for all Prosper python scripts. Easy extensions included to make life easy

##### Example

```
import prosper.common.prosper_logging as p_log

LogBuilder = p_log.ProsperLogger( 'log_name', 'desired/log/path', configuration_object, bool:debug_mode [optional]
)

LogBuilder.configure_discord_logger() # log ERROR/CRITICAL to Discord

if DEBUG: LogBuilder.configure_debug_logger()

logger = LogBuilder.get_logger()

class common.prosper_logging.DiscordWebhook
    Bases: object

        Helper object for parsing info and testing discord webhook credentials

        webhook_url
            address of webhook endpoint
                Type str

        serverid
            Discord 'guild' webhook is attached to
                Type int

        api_key (`str`
            uuid): unique ID for webhook

        api_keys (serverid, api_key)
            Load object with id/API pair

            Parameters
                • serverid (int) – Discord 'guild' webhook is attached to
                • (str (api_key)) – uuid: unique ID for webhook

        get_webhook_info()
            Ping Discord endpoint to make sure webhook is valid and working

        webhook (webhook_url)
            Load object with webhook_url

            Parameters webhook_url (str) – full webhook url given by Discord 'create webhook' func

        class common.prosper_logging.HackyDiscordHandler (webhook_obj,
                                                    alert_recipient=None)
            Bases: logging.Handler

            Custom logging.Handler for pushing messages to Discord
```

Should be able to push messages to any REST webhook with small adjustments

Stolen from [https://github.com/invernizzi/hiplogging/blob/master/hiplogging/\\_\\_init\\_\\_.py](https://github.com/invernizzi/hiplogging/blob/master/hiplogging/__init__.py)

Discord webhook API docs: <https://discordapp.com/developers/docs/resources/webhook>

**emit (record)**

required classmethod for logging to execute logging message

**send\_msg\_to\_webhook (message)**

separated Requests logic for easier testing

**Parameters** **message** (*str*) – actual logging string to be passed to REST endpoint

---

**Todo:**

- Requests.text/json return for better testing options
- 

**test (message)**

testing hook for exercising webhook directly

**class** common.prosper\_logging.**HackyHipChatHandler** (*webhook\_url*)

Bases: logging.Handler

custom logging.Handler for pushing messages to HipChat

**decorate (record)**

Build up HipChat specific values for log record

**Parameters** **record** (logging.record) – log message object

**Returns** params for POST request

**Return type** dict

**emit (record)**

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

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

**send\_msg\_to\_webhook (json\_payload, log\_msg)**

todo

**class** common.prosper\_logging.**HackySlackHandler** (*webhook\_url*)

Bases: logging.Handler

Custom logging.Handler for pushing messages to Slack

**decorate (record)**

add slack-specific flourishes to responses

<https://api.slack.com/docs/message-attachments>

**Parameters** **record** (logging.record) – message to log

**Returns** attachments object for reporting

**Return type** (dict)

**emit (record)**

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

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

```
send_msg_to_webhook(json_payload, log_msg)
push message out to webhook
```

#### Parameters

- **json\_payload** (dict) – preformatted payload a la <https://api.slack.com/docs/message-attachments>
- **log\_msg** (str) – actual log message

```
class common.prosper_logging.ProsperLogger(log_name, log_path, config_obj=<prosper.common.prosper_config.ProsperConfig object>, custom_args="")
```

Bases: object

One logger to rule them all. Build the right logger for your script in a few easy steps

#### logger

current built logger (use get\_logger() to fetch)

**Type** logging.Logger

#### log\_name

the name of the log/log\_object

**Type** str

#### log\_path

path for logfile. abspath > relpath

**Type** str

#### log\_info

list of ‘handler\_name @ log\_level’ for debug

**Type** list of str

#### log\_handlers

collection of all handlers attached (for testing)

**Type** list of logging.handlers

---

#### Todo:

- add args/local/global config priority management
- 

#### close\_handles()

cannot delete logs unless handles are closed (windows)

```
configure_debug_logger(log_level='DEBUG', log_format='[%(levelname)s:%(filename)s-%(funcName)s:(lineno)s%(custom_args)s] %(message)s', custom_args="")
```

debug logger for stdout messages. Replacement for print()

---

**Note:** Will try to overwrite minimum log level to enable requested log\_level

---

#### Parameters

- **log\_level** (str) – desired log level for handle <https://docs.python.org/3/library/logging.html#logging-levels>

- **log\_format** (*str*) – format for logging messages <https://docs.python.org/3/library/logging.html#logrecord-attributes>
- **custom\_args** (*str*) – special ID to include in messages

```
configure_default_logger(log_freq='midnight',           log_total=30,           log_level='INFO',
                        log_format='[%(asctime)s;%(levelname)s;%(filename)s;%(funcName)s;%(lineno)s{custom_
                        %(message)s', custom_args="")  
default logger that every Prosper script should use!!
```

#### Parameters

- **log\_freq** (*str*) – TimedRotatingFileHandle\_str – <https://docs.python.org/3/library/logging.handlers.html#timedrotatingfilehandler>
- **log\_total** (*int*) – how many log\_freq periods between log rotations
- **log\_level** (*str*) – minimum desired log level <https://docs.python.org/3/library/logging.html#logging-levels>
- **log\_format** (*str*) – format for logging messages <https://docs.python.org/3/library/logging.html#logrecord-attributes>
- **custom\_args** (*str*) – special ID to include in (ALL) messages

```
configure_discord_logger(discord_webhook=None,           discord_recipient=None,
                        log_level='ERROR',   log_format='[%(levelname)s:%(filename)s-
                        %(funcName)s:%(lineno)s{custom_args}\n%(message).1000s',
                        custom_args="")
```

logger for sending messages to Discord. Easy way to alert humans of issues

---

**Note:** Will try to overwrite minimum log level to enable requested log\_level Will warn and not attach hipchat logger if missing webhook key Learn more about webhooks: <https://support.discordapp.com/hc/en-us/articles/228383668-Intro-to-Webhooks>

---

#### Parameters

- **discord\_webhook** (*str*) – discord room webhook (full URL)
- **(str discord\_recipient)** – <@int>, optional): user/group to notify
- **log\_level** (*str*) – desired log level for handle <https://docs.python.org/3/library/logging.html#logging-levels>
- **log\_format** (*str*) – format for logging messages <https://docs.python.org/3/library/logging.html#logrecord-attributes>
- **custom\_args** (*str*) – special ID to include in messages

```
configure_hipchat_logger(hipchat_webhook=None,           log_level='ERROR',
                        log_format='[%(levelname)s:%(filename)s-
                        %(funcName)s:%(lineno)s{custom_args}\n%(message).1000s',
                        custom_args="")
```

logger for sending messages to HipChat. Easy way to alert humans of issues

---

**Note:** Will try to overwrite minimum log level to enable requested log\_level Will warn and not attach hipchat logger if missing webhook key Learn more about webhooks: <https://yak.crowdstrike.com/addons/byo>

---

### Parameters

- **hipchat\_webhook** (*str*) – slack bot webhook (full URL)
- **log\_level** (*str*) – desired log level for handle <https://docs.python.org/3/library/logging.html#logging-levels>
- **log\_format** (*str*) – format for logging messages <https://docs.python.org/3/library/logging.html#logrecord-attributes>
- **custom\_args** (*str*) – special ID to include in messages

```
configure_slack_logger(slack_webhook=None, log_level='ERROR',
log_format='%(message).1000s', custom_args="")
logger for sending messages to Slack. Easy way to alert humans of issues
```

---

**Note:** Will try to overwrite minimum log level to enable requested log\_level Will warn and not attach hipchat logger if missing webhook key Learn more about webhooks: <https://api.slack.com/docs/message-attachments>

---

### Parameters

- **slack\_webhook** (*str*) – slack bot webhook (full URL)
- **log\_level** (*str*) – desired log level for handle <https://docs.python.org/3/library/logging.html#logging-levels>
- **log\_format** (*str*) – format for logging messages <https://docs.python.org/3/library/logging.html#logrecord-attributes>
- **custom\_args** (*str*) – special ID to include in messages

```
get_logger()
return the logger for the user
```

```
class common.prosper_logging.ReportingFormats
Bases: enum.Enum

Enum for storing handy log formats

DEFAULT = '[% (asctime) s;% (levelname) s;% (filename) s;% (funcName) s;% (lineno) s{custom_args}'
PRETTY_PRINT = '[% (levelname) s:% (filename) s--% (funcName) s:% (lineno) s{custom_args}] \n(%'
SLACK_PRINT = '% (message) .1000s'

STDOUT = '[% (levelname) s:% (filename) s--% (funcName) s:% (lineno) s{custom_args}] % (message

common.prosper_logging.test_logpath(log_path, debug_mode=False)
Tests if logger has access to given path and sets up directories
```

---

**Note:** Should always yield a valid path. May default to script directory Will throw warnings.RuntimeWarning if permissions do not allow file write at path

---

### Parameters

- **log\_path** (*str*) – path to desired logfile. Abspath > relpath
- **debug\_mode** (*bool*) – way to make debug easier by forcing path to local

**Returns**

path to log

if path exists or can be created, will return log\_path else returns ‘.’ as “local path only” response

**Return type** str

#### 4.1.6 common.prosper\_utilities module

#### 4.1.7 common.prosper\_version module

prosper\_version.py: utilities to help parse current version information

Props to ccpgames/setuputils for framework

`common.prosper_version.get_version(here_path, default_version='0.0.0')`  
tries to resolve version number

**Parameters**

- `here_path` (str) – path to project local dir
- `default_version` (str) – what version to return if all else fails

**Returns** semantic\_version information for library

**Return type** str

#### 4.1.8 Module contents



# CHAPTER 5

---

## Indices and tables

---

- genindex
- modindex
- search



---

## Python Module Index

---

### C

common, 25  
common.exceptions, 15  
common.prosper\_cli, 16  
common.prosper\_config, 17  
common.prosper\_logging, 20  
common.prosper\_version, 25



---

## Index

---

### A

api\_keys () (*common.prosper\_logging.DiscordWebhook* debug method), 20

### C

check\_value () (*in module common.prosper\_config*), 18  
close\_handles () (*common.prosper\_logging.ProsperLogger* method), 22  
common (*module*), 25  
common.exceptions (*module*), 15  
common.prosper\_cli (*module*), 16  
common.prosper\_config (*module*), 17  
common.prosper\_logging (*module*), 20  
common.prosper\_version (*module*), 25  
config (*common.prosper\_cli.ProsperApplication* attribute), 16  
config\_filename (*common.prosper\_config.ProsperConfig* attribute), 17  
config\_path (*common.prosper\_cli.ProsperTESTApplication* attribute), 17  
configure\_debug\_logger () (*common.prosper\_logging.ProsperLogger* method), 22  
configure\_default\_logger () (*common.prosper\_logging.ProsperLogger* method), 23  
configure\_discord\_logger () (*common.prosper\_logging.ProsperLogger* method), 23  
configure\_hipchat\_logger () (*common.prosper\_logging.ProsperLogger* method), 23  
configure\_slack\_logger () (*common.prosper\_logging.ProsperLogger* method), 24

### D

debug (*common.prosper\_cli.ProsperApplication* attribute), 16  
decorate () (*common.prosper\_logging.HackyHipChatHandler* method), 21  
decorate () (*common.prosper\_logging.HackySlackHandler* method), 21  
DEFAULT (*common.prosper\_logging.ReportingFormats* attribute), 24  
DiscordWebhook (*class in common.prosper\_logging*), 20  
dump\_config () (*common.prosper\_cli.ProsperApplication* method), 16

### E

emit () (*common.prosper\_logging.HackyDiscordHandler* method), 21  
emit () (*common.prosper\_logging.HackyHipChatHandler* method), 21  
emit () (*common.prosper\_logging.HackySlackHandler* method), 21

### F

FlaskLauncher (*class in common.prosper\_cli*), 16

### G

get () (*common.prosper\_config.ProsperConfig* method), 18  
get\_configs () (*in module common.prosper\_config*), 19  
get\_host () (*common.prosper\_cli.FlaskLauncher* method), 16  
get\_local\_config\_filepath () (*in module common.prosper\_config*), 19  
get\_logger () (*common.prosper\_logging.ProsperLogger* method), 24

get\_option() (com-  
*mon.prosper\_config.ProsperConfig* method), 18  
get\_value\_from\_environment() (in module *common.prosper\_config*), 19  
get\_version() (in module *mon.prosper\_version*), 25  
get\_webhook\_info() (com-  
*mon.prosper\_logging.DiscordWebhook*  
method), 20  
global\_config (com-  
*mon.prosper\_config.ProsperConfig* attribute), 17

**H**

HackyDiscordHandler (class in *com-  
mon.prosper\_logging*), 20  
HackyHipChatHandler (class in *com-  
mon.prosper\_logging*), 21  
HackySlackHandler (class in *com-  
mon.prosper\_logging*), 21  
HERE (*common.prosper\_cli.ProsperTESTApplication* attribute), 17

**L**

load\_secrets() (com-  
*mon.prosper\_cli.ProsperApplication* method), 16  
local\_config (com-  
*mon.prosper\_config.ProsperConfig* attribute), 17  
local\_config\_filename (com-  
*mon.prosper\_config.ProsperConfig* attribute), 17  
log\_handlers (com-  
*mon.prosper\_logging.ProsperLogger* attribute), 22  
log\_info (*common.prosper\_logging.ProsperLogger* attribute), 22  
log\_name (*common.prosper\_logging.ProsperLogger* attribute), 22  
log\_path (*common.prosper\_logging.ProsperLogger* attribute), 22  
logger (*common.prosper\_cli.ProsperApplication* attribute), 16  
logger (*common.prosper\_config.ProsperConfig* attribute), 18  
logger (*common.prosper\_logging.ProsperLogger* attribute), 22

**M**

main() (*common.prosper\_cli.ProsperTESTApplication* method), 17

**N**

notify\_launch() (com-  
*mon.prosper\_cli.FlaskLauncher* method), 16

**O**

override\_config() (com-  
*mon.prosper\_cli.ProsperApplication* method), 16

**P**

port (*common.prosper\_cli.FlaskLauncher* attribute), 16  
PRETTY\_PRINT (com-  
*mon.prosper\_logging.ReportingFormats* attribute), 24  
PROGNAME (*common.prosper\_cli.ProsperTESTApplication* attribute), 17  
ProsperApplication (class in *com-  
mon.prosper\_cli*), 16  
ProsperCommonException, 15  
ProsperCommonWarning, 15  
ProsperConfig (*class in common.prosper\_config*), 17  
ProsperDefaultVersionWarning, 15  
ProsperLogger (*class in common.prosper\_logging*), 22  
ProsperLoggerWarning, 15  
ProsperTESTApplication (class in *com-  
mon.prosper\_cli*), 17  
ProsperVersionException, 15  
ProsperVersionTestModeWarning, 15  
ProsperVersionWarning, 15

**R**

read\_config() (in module *common.prosper\_config*), 19  
render\_secrets() (in module *mon.prosper\_config*), 19  
ReportingFormats (class in *com-  
mon.prosper\_logging*), 24

**S**

send\_msg\_to\_webhook() (com-  
*mon.prosper\_logging.HackyDiscordHandler* method), 21  
send\_msg\_to\_webhook() (com-  
*mon.prosper\_logging.HackyHipChatHandler* method), 21  
send\_msg\_to\_webhook() (com-  
*mon.prosper\_logging.HackySlackHandler* method), 21  
serverid (*common.prosper\_logging.DiscordWebhook* attribute), 20  
SLACK\_PRINT (*common.prosper\_logging.ReportingFormats* attribute), 24

STDOUT (*common.prosper\_logging.ReportingFormats* attribute), 24

## T

test () (*common.prosper\_logging.HackyDiscordHandler* method), 21  
test\_logpath () (in module *common.prosper\_logging*), 24  
threaded (*common.prosper\_cli.FlaskLauncher* attribute), 16

## V

verbose (*common.prosper\_cli.ProsperApplication* attribute), 17  
VERSION (*common.prosper\_cli.ProsperTESTApplication* attribute), 17

## W

webhook () (*common.prosper\_logging.DiscordWebhook* method), 20  
webhook\_url (*common.prosper\_logging.DiscordWebhook* attribute), 20  
WebhookCreateFailed, 16  
WebhookFailedEmitWarning, 16  
workers (*common.prosper\_cli.FlaskLauncher* attribute), 16