ProsperCommon Documentation

Release 1.0.2

John Purcell

Jul 10, 2018

Contents:

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_loging

LogBuilder = p_loging.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()

```
def configure_default_logger(
    log_freq:str,
    log_total:int,
    log_level:log_level_str,
    log_format:log_format_str,
    debug_mode:bool
):
```

- **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()

```
def configure_debug_logger(
    log_level:log_level_str,
    log_format:log_format_str,
    debug_mode:bool
):
```

- **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()

```
def configure_discord_logger(
    discord_webhook:url_str,
    discord_recipient:'<@int>'_discord_id_str,
    log_level:log_level_str,
    log_format:log_format_str,
    debug_mode:bool
):
```

- **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()

```
def configure_slack_logger(
    slack_webhook:url_str,
    log_level:log_level_str,
    log_format:log_format_str
    debug_mode:bool
):
```

- **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 ccpgames/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 confilcts.

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, PropsperCLI 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(),
```

(continues on next page)

(continued from previous page)

```
            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 preferrable 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 libaries

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/lates

PROGNAME = 'CLITEST'

VERSION = '0.0.0'

config_path = '/home/docs/checkouts/readthedocs.org/user_builds/prospercommon/checkout

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
configparser.ConfigParser

local_config
configparser.ConfigParser

config_filename
str – filename of global/tracked/default .cfg file

local_config_filename
str – filename for local/custom .cfg file

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 = <logging.Logger object>

common.prosper_config.**check_value** (*config*, *section*, *option*, *jinja_pattern=re.compile('.*{{\\S*}}.*')*)
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=<logging.Logger object>*)
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=<logging.Logger object>*)
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
str – address of webhook endpoint

serverid
int – Discord 'guild' webhook is attached to

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

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
logging.Logger – current built logger (use get_logger() to fetch)

log_name
str – the name of the log/log_object

log_path
str – path for logfile. abspath > relpath

log_info
list of str – list of 'handler_name @ log_level' for debug

log_handlers
list of logging.handlers – collection of all handlers attached (for testing)

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}] %(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

Index