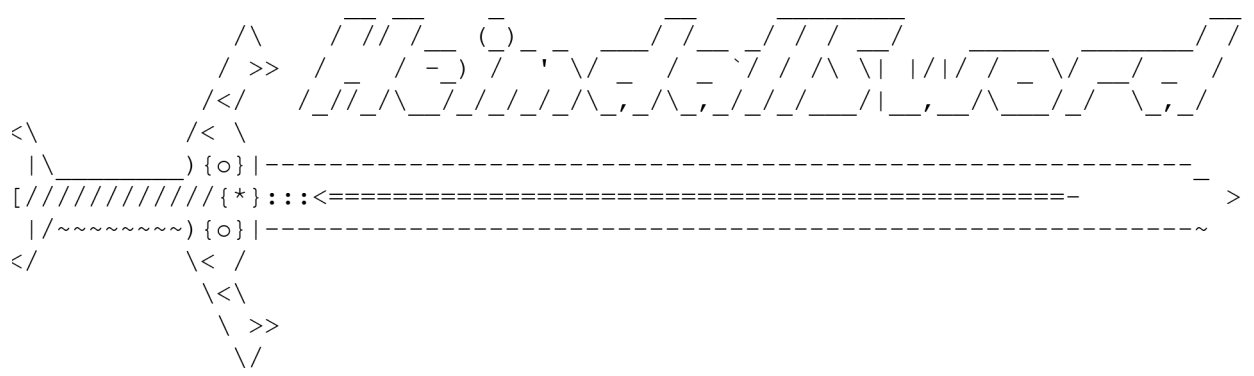
heimdallsword

Release 0.0.2

eldiablo

Jun 29, 2022

CONTENTS:



0.0.2 by eldiablo

A command line tool and a Python module used for sending emails to multiple recipients using one or multiple sender accounts with customizable email templates.

HeimdallSword is a command line tool and a Python module used for sending emails to multiple recipients using one or multiple sender accounts with customizable email templates.

From the command line, emails can be sent by simply providing the necessary files that contain the list of senders, recipients and the content.

As a Python module, **HeimdallSword** provides several modules which allows you send emails given the necessary requirements.

CHAPTER

ONE

BUILD

This section covers several build aspects of the **HeimdallSword** project.

1.1 Required Debian Packages

The following packages are required:

Dependency	Version
debhelper	12.10ubuntu1
dh-python	4.20191017ubuntu7
make	4.2.1
python3-all	3.8.2-0ubuntu2
python3-venv	3.8.2-0ubuntu2
python3-setuptools	45.2.0-1

Copy and paste the following command on your terminal:

```
sudo apt install debhelper dh-python make python3-all python3-venv python3-setuptools
```

1.2 Required PyPi Packages

The following PyPi packages are required:

Dependency	Version	Purpose
build	0.7.0	Used to build Python distribution package
stdeb	0.10.0	Used to build Debian source package

Copy and paste the following command on your terminal:

```
pip install build stdeb
```

1.3 Optional PyPi Packages

The following PyPi packages are optional to install:

Dependency	Version	Purpose
flake8	4.0.1	Used for checking code style
sphinx_rtd_theme	4.4.0	Used for generating documentation

Copy and paste the following command on your terminal:

```
pip install flake8 sphinx_rtd_theme
```

1.4 Build Targets

Once all of the required dependencies have been installed, **HeimdallSword** provides several build targets that make it easier to build and deploy.

To view the build targets supported, run the command *make help* as such:

```
$ make help

The following build targets are available:

build-pip:     Generates a PIP distributions packages (ie: *.whl and *.tar.gz)
build-deb:     Generates a Debian Linux distribution package (ie: *.deb)
install-pip:   Builds and installs PIP distribution package
install-deb:   Builds and installs Debian distribution package
uninstall-pip: Uninstalls PIP distribution package
uninstall-deb: Uninstalls Debian distribution package
html:          Builds the HTML version of the docs
clean:         Removes any automatically generated files
```

CHAPTER

TWO

INSTALLATION

This section provides several methods for installing **HeimdallSword**.

2.1 From the Source

HeimdallSword can be build and deployed directly from the source. It can be obtained as follows:

```
$ git clone https://github.com/rwprimitives/heimdallsword.git
```

2.2 Installing from PyPi

It is highly recommended that **HeimdallSword** be installed using *pip* to ensure that the latest version is being used.

To install simply run:

```
$ pip install heimdallsword
```

CHAPTER

THREE

CONTRIBUTIONS

Contributions to the project can be made by doing one of the following:

1. Check for open issues before submitting a feature or bug.
2. Create a new issue to start a discussion around a new feature or a bug.

CHAPTER

FOUR

LICENSE

CHAPTER

FIVE

API GUIDE

5.1 Config Module

This module contains configuration information needed for the the entire project.

class heimdallsword.data.config.**Config**

Bases: object

The Config class contains attributes used to describe the flow of operation for sending emails as well as Constants used as default values for most attributes.

DEFAULT_CONTENT_DIR = 'content'

Content directory default name

DEFAULT_DELAY = 100

Default delay value in milliseconds used between each email sent

DEFAULT_LOG_FILE_PATH = './heimdallsword.log'

Default log file name and path

DEFAULT_METRICS_DELAY = 120

Default metrics delay value in seconds used to wait after sending an email before logging into the sender's account and retrieving bounced emails

DEFAULT_METRICS_FILE_PATH = './metrics.txt'

Default metrics file name and path

DEFAULT_POP3_PORT = 995

Default POP3 SSL port

DEFAULT_RECIPIENTS_FILE = 'recipients.txt'

Recipients file default name

DEFAULT_SENDERS_FILE = 'senders.txt'

Senders file default name

DEFAULT_SMTP_PORT = 587

Default SMTP port

5.2 Metrics Module

This module contains metrics generated when sending emails.

class heimdallsword.data.metrics.**Metrics**(*metrics_file=None*)

Bases: object

The *heimdallsword.data.metrics.Metrics* class is used to track various values when sending emails. These values are used to perform several calculations to assess, compare and track performance of the entirety email sending operation.

The following values are tracked in order to determine the accuracy of emails sent:

start_time	the timestamp before the operation starts
stop_time	the timestamp after the operation ended
num_of_senders	the number of sender accounts
num_of_recipients	the number of recipient accounts
num_of_emails_not_delivered	the number of emails not delivered
num_of_emails_delivered	the number of emails successfully delivered
num_of_emails_failed_delivery	the number of emails that failed to be delivered
num_of_recipients_rejected	the number of recipients rejected by the recipient's SMTP server due to an invalid recipient
num_of_senders_rejected	the number of senders rejected by the recipient's SMTP server
num_of_emails_failed_delivery_format	the number of emails failed to deliver due to an invalid format
num_of_emails_disconnected	the number of emails that weren't delivered due to a failed connection with the sender's SMTP server

Parameters
metrics_file – a file path including a file name for the metrics file

Type
str

activate_start_time()

Set the *start_time* to a timestamp generated from datetime.datetime.now().

activate_stop_time()

Set the *stop_time* to a timestamp generated from datetime.datetime.now().

get_current_delivery_rate()

Get the current delivery rate. This is calculated based on the total number of emails delivered and the total number of recipients.

Returns
the delivery rate

Return type
float

get_current_fail_rate()

Get the current fail rate. This is calculated based on the following values:

- num_of_emails_failed_delivery
- num_of_recipients_rejected

- num_of_senders_rejected
- num_of_emails_disconnected
- num_of_emails_failed_delivery_format
- num_of_recipients

Returns
the failure rate

Return type
float

`get_disconnected_count()`

Get the number of emails that weren't delivered due to a failed connection with the sender's SMTP server.

This method is thread-safe.

Returns
the number of emails failed to send due to sender's connection failure with SMTP server

Return type
int

`get_elapsed_time()`

Calculate the time difference between the *start_time* and *stop_time*.

Returns
a string representation of the time difference

Return type
str

`get_emails_delivered_count()`

Get the number of emails delivered.

This method is thread-safe.

Returns
the number of emails delivered

Return type
int

`get_emails_failed_delivery_count()`

Get the number of emails that failed to be delivered.

This method is thread-safe.

Returns
the number of emails that failed to be delivered

Return type
int

`get_emails_not_delivered_count()`

Get the number of emails not delivered.

This method is thread-safe.

Returns
the number of emails not delivered

Return type
int

`get_failed_delivery_format_count()`

Get the number of emails failed to deliver due to an invalid format.

This method is thread-safe.

Returns
the number of emails failed to delivery due to invalid format

Return type
int

`get_num_of_recipients()`

Get the number of recipients.

Returns
the number of recipients

Return type
int

`get_num_of_senders()`

Get the number of senders.

Returns
the number of senders

Return type
int

`get_recipient_rejected_count()`

Get the number of recipients rejected by the recipient's SMTP server due to an invalid recipient.

This method is thread-safe.

Returns
the number of rejected recipients

Return type
int

`get_senders_rejected_count()`

Get the number of senders rejected by the recipient's SMTP server.

This method is thread-safe.

Returns
the number of rejected senders

Return type
int

`get_start_time(dt_format='%m/%d/%Y %H:%M:%S.%f')`

Get the start time.

Parameters
dt_format – a string containing format codes for date and time

Type
str

Returns
a tuple containg the start time as a timestamp and a formatted string. Zero and N/A may be returned if *start_time* is zero

Return type
tuple

`get_stop_time`(*dt_format='%m/%d/%Y %H:%M:%S.%f'*)

Get the stop time.

Parameters
dt_format – a string containing format codes for date and time

Type
str

Returns
a tuple containg the stop time as a timestamp and a formatted string. Zero and N/A may be returned if *stop_time* is zero

Return type
tuple

`increment_disconnected_count`()

Increment the number of emails that weren't delivered due to a failed connection with the sender's SMTP server counter by one.

This method is thread-safe.

`increment_emails_delivered_count`()

Increment the number of emails delivered counter by one.

This method is thread-safe.

`increment_emails_failed_delivery_count`()

Increment the number of emails that failed to e delivered counter by one.

This method is thread-safe.

`increment_emails_not_delivered_count`()

Increment the number of emails not delivered counter by one.

This method is thread-safe.

`increment_failed_delivery_format_count`()

Increment the number of emails failed to deliver due to invalid format counter by one.

This method is thread-safe.

`increment_recipient_rejected_count`()

Increment the number of recipients rejected counter by one.

This method is thread-safe.

`increment_senders_rejected_count`()

Increment the number of senders rejected counter by one.

This method is thread-safe.

`save_metrics`(*is_json=False*, *is_json_prettyprint=True*)

Save the metrics data to file defined by *metrics_file*. The metrics data by default is saved as key-value pairs, however *is_json* can be set to True to save the data in JSON format.

Parameters

- **is_json** – True to write the metrics in JSON format, False to write the metrics as key-value pairs. Default value is False
- **is_json_prettyprint** – True to enable pretty print JSON data if *is_json* is enabled

Type
bool

Type
bool

set_metrics_file(*metrics_file*)

Set the file path and file name to store the metrics data.

This method will throw an *IOError* if nothing is passed.

Parameters
metrics_file – a file path including a file name for the metrics file

Type
str

set_number_of_recipients(*num_of_recipients*)

Set the number of recipients.

Parameters
num_of_recipients – the number of recipients

Type
int

set_number_of_senders(*num_of_senders*)

Set the number of senders.

Parameters
num_of_senders – the number of senders

Type
int

5.3 Client Module

This module is a thread-safe email client.

class heimdallsword.dispatcher.client.**EmailClient**(*sender*, *metrics_delay=120*)

Bases: object

The *heimdallsword.dispatcher.client.EmailClient* class serves as an email client used to manage an SMTP connection and send emails to any given recipients. This is a thread-safe class.

Parameters

- **sender** – a reference to a *heimdallsword.models.sender.Sender* object that contains the email address and password to establish a connection with it's SMTP server which will be used to send emails to any recipient
- **metrics_delay** – the number of seconds to wait between sending an email and checking the sender inbox for bounced emails

Type
heimdallsword.models.sender.Sender

Type
int

`acquire_lock()`

Acquires the module-level I/O thread lock.

`get_lock()`

Get a reference to the module-level I/O thread lock.

Returns
A reference to the lock

Return type
`threading.Lock`

`is_connection_active()`

Checks to see if the connection with the SMTP server is still alive by sending an SMTP 'noop' command which doesn't do anything.

This method is thread-safe.

Returns
True on successful connection, False otherwise

Return type
bool

`release_lock()`

Releases the module-level I/O thread lock.

`send(`*recipient*`)`

Attempts to send an email to a given *recipient*. This method rethrows any exception thrown by the `smtplib.SMTP` module.

This method is thread-safe.

When an exception is caught, any error codes generated by the SMTP server are stored in the `heimdallsword.models.recipient.Recipient.delivery_error_code` attribute, error messages are stored in the `heimdallsword.models.recipient.Recipient.delivery_error_message` attribute, and the `heimdallsword.models.recipient.Recipient.delivery_state` attribute is set with a constant type from *heimdallsword.models.recipient.DeliveryState* class.

This method returns *heimdallsword.models.recipient.DeliveryState.SUCCESSFUL_DELIVERY*, otherwise it returns any of the other type of failed constants to describe as much as possible why it failed to send the email.

Parameters
recipient – the recipient object

Type
heimdallsword.models.recipient.Recipient

Returns
heimdallsword.models.recipient.DeliveryState.SUCCESSFUL_DELIVERY on successful delivery of an email

Return type
heimdallsword.models.recipient.DeliveryState

`terminate_session()`

Closes an existing SMTP connection.

This method is thread-safe.

NOTE: This method will catch `smtplib.SMTPServerDisconnected` exception that is thrown when calling `SMTP.quit()` and it is ignored as it has no effect to the process if the session couldn't be terminated.

`test_connection()`

Attempts to establish a connection between the sender and it's SMTP server.

This method is thread-safe.

Returns
True on successful connection, False otherwise

Return type
bool

5.4 Orchestrator Module

This module serves as an Email client which establishes and maintains a connection with the SMTP server in order to send emails and check the status of the emails sent in a multi-threaded fashion

`class heimdallsword.dispatcher.orchestrator.Orchestrator(config, metrics, worker_count=None)`

Bases: `object`

The *heimdallsword.dispatcher.orchestrator.Orchestrator* class serves as the organizer for the entirity of the email sending operation.

Emails can be sent concurrently or sequentially based on the number of working threads defined in *worker_count*. By default, *worker_count* is set by taking the number of processors on the machine and multiplying it by 5. This method of calculation is used in the `concurrent.futures.ThreadPoolExecutor` module when *max_workers* is set to None.

As emails are sent, metrics are gathered to help calculate success or failure rates. The *heimdallsword.dispatcher.orchestrator.Orchestrator* class allows for subscribers to register and receive notifications when metrics are updated.

A `logging` object can be passed by calling the method *set_logger()* if logging information is desired.

Parameters

- **config** – a reference to a *heimdallsword.data.config.Config* object that contains the configuration which describes the flow of operations for sending emails
- **metrics** – a reference to a *heimdallsword.data.metrics.Metrics* object used to manage data that is tracked for the purpose of generating metrics
- **worker_count** – the numer of worker threads to use for sending emails. Default value varies per machine

Type
heimdallsword.data.config.Config

Type
heimdallsword.data.metrics.Metrics

Type
int

add_subscriber(*subscriber*)

Add a subscriber to receive metrics update notifications.

All subscribers must inherit the base class *heimdallsword.dispatcher.subscriber.Subscriber* and implement the *heimdallsword.dispatcher.subscriber.Subscriber.update_metrics()* method.

Parameters
subscriber – a subscriber

Type
heimdallsword.dispatcher.subscriber.Subscriber

notify_subscribers(*metrics*)

Notify subscribers of metrics updates.

Parameters
metrics – a reference to a *heimdallsword.data.metrics.Metrics* object used to manage data that is tracked for the purpose of generating metrics

Type
heimdallsword.data.metrics.Metrics

remove_subscriber(*subscriber*)

Remove a subscriber to stop receiving metrics update notifications.

This method will throw a *ValueError* exception if the suscriber is not in the list.

Parameters
subscriber – the subscriber to remove

Type
heimdallsword.dispatcher.subscriber.Subscriber

set_config(*config*)

Set the configuration.

This method will return *IOError* exception if config is not provided.

Parameters
config – a reference to a *heimdallsword.data.config.Config* object that contains the configuration which describes the flow of operations for sending emails.

Type
heimdallsword.data.config.Config

set_content(*senders*, *recipients*)

Set the sender and recipient content.

This method will return *IOError* exception if a sender or recipient is not provided.

Parameters

- **senders** – a reference to a *heimdallsword.models.sender.Sender* class that contains one or more senders
- **recipients** – a reference to a *heimdallsword.models.recipient.Recipient* class that contains one or more recipients

Type
heimdallsword.models.sender.Sender

Type
heimdallsword.models.sender.Sender

`set_logger`(*logger*)

Set the logger based `logging.Logger` to use for logging information.

Parameters
logger – the logger object to call for logging information

Type
`logging.Logger`

`start`()

Starts the process of sending emails using *n* number of worker threads. Each sender is paired with a recipient and are assigned a worker thread. Once a worker thread finishes the operation, it notifies all subscribers of updates in the metrics data.

NOTE: Worker threads will stay active even after all emails are sent. The worker threads will only terminate once the program terminates.

5.5 Subscriber Module

This module is an Observer interface which defines update methods.

`class heimdallsword.dispatcher.subscriber.Subscriber`

Bases: `object`

The *heimdallsword.dispatcher.subscriber.Subscriber* is an **Observer** interface used for the purpose of receiving *heimdallsword.data.metrics.Metrics* as the *heimdallsword.dispatcher.orchestrator.Orchestrator* publishes new updates.

`abstract update_metrics`(*metrics:* Metrics) → None

Receive metrics update from Orchestrator.

Parameters
metrics – a reference to a *heimdallsword.data.metrics.Metrics* class

Type
heimdallsword.data.metrics.Metrics

5.6 Renderer Module

This module contains a graphics renderer based on :py:mod:curses to display metrics and logging information on the terminal.

`class heimdallsword.graphics.renderer.CliRenderer`(*config*, *metrics*)

Bases: *Subscriber*

The *heimdallsword.graphics.renderer.CliRenderer* class produces a beautifully designed command line graphical interface which provides live metrics updates as emails are sent as well as logging information throughout the enterity of the operation.

Parameters

- **config** – a reference to a *heimdallsword.data.config.Config* object that contains the configuration which describes the flow of operations for sending emails

- **metrics** – a reference to a *heimdallsword.data.metrics.Metrics* object used to manage data that is tracked for the purpose of generating metrics

Type

py:class
heimdallsword.data.config.Config

Type
heimdallsword.data.metrics.Metrics

init()

Initialize and display the command line graphical interface.

Raises
IOError - Not enough space available to render graphics panel

run()

Start monitoring for key strokes and window size changes.

terminate()

Stop monitoring for key strokes and window size changes. Terminate the command line graphical interface and reset the terminal back to it's original state.

update_log(*record*)

Display logging information on the log window.

This method will ignore metric logs generated by the Orchestrator since the metrics stats are updated as notifications are received from then Orchestrator. Hence no need to display the same result in the log window.

This method is used as a callback in the *heimdallsword.log.handler.LogHandler* class. That way, whenever a log record is generated, a `logging.LogRecord` object is passed to this method and it can be displayed in the log window.

Parameters
record – a `logging.LogRecord` instance representing an event logged

Type
`logging.LogRecord`

update_metrics(*metrics:* Metrics)

Receive metrics update from Orchestrator and update data on screen.

Parameters
metrics – a reference to a *heimdallsword.data.metrics.Metrics* class

Type
heimdallsword.data.metrics.Metrics

5.7 CLI Log Module

This module provides a simple way of logging output to the terminal.

class heimdallsword.log.cli_log.**Colors**

Bases: `object`

A class used for ANSI color codes in a POSIX terminal.

BLUE = '\x1b[94m'

Blue color.

ENDC = '\x1b[0m'

End color tag.

GREEN = '\x1b[92m'

Green color.

RED = '\x1b[31m'

Red color.

WHITE = '\x1b[37m'

White color.

YELLOW = '\x1b[93m'

Yellow color.

heimdallsword.log.cli_log.**get_log_message**(*record*)

Get a string representation of a log `logging.LogRecord`.

Parameters
record – a `logging.LogRecord` instance representing an event logged

Type
`logging.LogRecord`

Returns
a string repesentation of a log record

Return type
str

heimdallsword.log.cli_log.**log**(*record*)

Display a the content of a `logging.LogRecord` on a terminal using ANSI color codes for the different log levels.

This method is used as a callback in the *heimdallsword.log.handler.LogHandler* class. That way, whenever a log record is generated, a `logging.LogRecord` object is passed to this method and a color is used based on the log level and displayed on the terminal window.

The following colors are assigned to a specific log level:

INFO = *heimdallsword.log.cli_log.Colors.GREEN*

WARNING = *heimdallsword.log.cli_log.Colors.YELLOW*

ERROR = *heimdallsword.log.cli_log.Colors.RED*

Parameters
record – a `logging.LogRecord` instance representing an event logged

Type
`logging.LogRecord`

5.8 Logging Handler Module

This module inherits from the logging.Handler class and provides a way to set a callback method to invoke when log records are created.

class heimdallsword.log.handler.**LogHandler**(*stream=None*)

Bases: StreamHandler

A custom log handler used to invoke a callback method whenever a log record is created.

emit(*record*)

Emit a record.

If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using traceback.print_exception and appended to the stream. If the stream has an 'encoding' attribute, it is used to determine how to do the output to the stream.

set_callback(*callback*)

Set the callback method to invoke when a log record is created.

Parameters
callback – callback method

Type
logging.LogRecord

5.9 Message Module

This module contains the content defined in a message file.

class heimdallsword.models.message.**Message**

Bases: object

The *heimdallsword.models.message.Message* class represents the content required when composing an email.

HTML = 'html'

The MIME sub content type that represents *text/html*.

PLAIN = 'plain'

The MIME sub content type that represents *text/plain*.

get_body()

Get the body of the email.

Returns
the body of the email

Return type
str

get_content_type()

Get the content type.

Returns
the content type

Return type
str

`get_subject()`

Get the subject line.

Returns
the subject line

Return type
str

`set_body(body)`

Set the email body which contains the message.

Parameters
body – the body of the email

Type
str

`set_content_type(content_type)`

Set the content type. The type of content is the MIME sub content type. This can only be either "plain" or "html".

Parameters
content_type – the type of content

Type
str

`set_subject(subject)`

Set the subject line which summarizes the email.

Parameters
subject – the subject line

Type
str

5.10 Recipient Module

This module contains information about the recipient as well as delivery status information.

`class heimdallsword.models.recipient.DeliveryState`

Bases: `object`

The *DeliveryState* class provides constants that define the delivery status of a recipient.

`DISCONNECTED = 3`

Message was not delivered due to loss of connectivity with SMTP server. This state follows the exception SMTPServerDisconnected

`FAILED_DELIVERY = 2`

Message failed to reach recipient This state follows the exception SMTPHeloError, SMTPDataError, SMTPResponseException

INVALID_FORMAT = 4

Mail parameters provided are not supported, i,e., SMTPUTF8 This state follows the exception SMTPNotSupportedError, ValueError

NOT_DELIVERED = 1

Message has not been sent to recipient

RECIPIENT_REJECTED = 5

Mail server rejected recipient This state follows the exception SMTPRecipientsRefused

SENDER_REJECTED = 6

Mail server rejected sender email This state follows the exception SMTPSenderRefused

SUCCESSFUL_DELIVERY = 0

Message was successfully delivered to recipient

class heimdallsword.models.recipient.**Recipient**

Bases: `object`

The *Recipient* class represents a recipient.

add_custom_tag(*key*, *value*)

Add a custom tag given a key-value pair.

Parameters

- **key** – the key
- **value** – the value

Type
str

Type
str

get_all_custom_tags()

Get the `Dictionary` that contains a list of custom tags associated with a message template file.

Returns
a list of custom tags

Return type
`Dictionary`

get_custom_tag(*key*)

Get the value of a custom tag given a key.

Parameters
key – the key of the tag

Type
str

Returns
the value

Return type
str

`get_delivery_error_code()`

Get the SMTP error code provided by the SMTP server.

Returns
the SMTP error code

Return type
int

`get_delivery_error_message()`

Get the SMTP error message provided by the SMTP server.

Returns
the SMTP error message

Return type
str

`get_delivery_state()`

Get the state of the delivery of the email to the recipient.

Returns
the state of delivery

Return type
heimdallsword.models.recipient.DeliveryState

`get_email()`

Get the email address of the recipient.

Returns
the recipient's email address

Return type
str

`get_email_domain()`

Get the email domain URL.

Returns
the domain URL of the email address

Return type
str

`get_email_username()`

Get the email username.

Returns
the email username

Return type
str

`get_message()`

Get the `Message` object associated with this recipient.

Returns
the `Message`

Return type
`Message`

`get_message_filename()`

Get the filename of the message file.

Returns
the filename of the message

Return type
str

`get_msg_id()`

Get the RFC 2822-compliant Message-ID header string assigned to the recipient.

Returns
RFC 2822-compliant Message-ID header string

Return type
str

`get_sending_timestamp()`

Get the timestamp before the email is sent.

Returns
the sending timestamp

Return type
str

`get_sent_timestamp()`

Get the timestamp after the email was successfully sent.

Returns
the sent timestamp

Return type
str

`set_custom_tags(tags)`

Set a `Dictionary` object which contains a list of key-value pairs defined in the message content.

Parameters
tags – the custom tags

Type
dict

Raises
IOError - tags must be of dictionary type

`set_delivery_error_code(delivery_error_code)`

Set the SMTP error code provided by the SMTP server.

Parameters
delivery_error_code – the SMTP error code

Type
int

`set_delivery_error_message(delivery_error_message)`

Set the SMTP error message provided by the SMTP server.

Parameters
delivery_error_message – the SMTP error message

Type
str

`set_delivery_state`(*delivery_state*)

Set the state of the delivery of the email to the recipient.

Parameters
delivery_state – the delivery state

Type
heimdallsword.models.recipient.DeliveryState

`set_email`(*email*)

Set the email address of the recipient.

Parameters
email – the recipient's email address

Type
str

Raises
IOError - Invalid recipient email

`set_message`(*message*)

Set a `Message` object based on the content in the message file (`Recipient.message_filename`).

Parameters
message – the `Message`

Type
`Message`

`set_message_filename`(*message_filename*)

Set the filename which contains the message to send to the recipient.

Parameters
message_filename – the filename of the message file

Type
str

`set_msg_id`(*msg_id*)

Set the RFC 2822-compliant Message-ID header string used for tracking an email sent.

Parameters
msg_id – the message ID assigned

Type
str

`set_sending_timestamp`(*sending_timestamp*)

Set the timestamp before the email is sent.

Parameters
sending_timestamp – the timestamp before the email is sent

Type
str

set_sent_timestamp(*sent_timestamp*)

Set the timestamp after the email was successfully sent.

Parameters
sent_timestamp – the timestamp after the email was successfully sent

Type
str

5.11 Sender Module

This module contains information about the sender account.

class heimdallsword.models.sender.**Sender**

Bases: object

The *Sender* class represents a sender.

Parameters

- **email** – the email address of the sender
- **password** – the password of the sender email account to authenticate with the SMTP server
- **smtp_port** – the port number used to establish an SMTP session. Default port is 587
- **smtp_url** – the sender's SMTP URL to authenticate. Default is the domain of the sender's email
- **pop3_port** – the port number used to establish a POP3 session. Default port is 995
- **pop3_url** – the sender's POP3 URL to authenticate. Default is the domain of the sender's email

Type
str

Type
str

Type
int

Type
str

Type
int

Type
str

get_email()

Get the email address of the sender.

Returns
the sender's email

Return type
str

`get_password()`

Get the password of the sender email account to authenticate with the SMTP server.

Returns
the sender's password

Return type
str

`get_pop3_port()`

Get the port number used to establish a POP3 session.

Returns
the port number used to establish a POP3 session.

Return type
int

`get_pop3_url()`

Get the sender's POP3 URL to authenticate. If no POP3 URL was specified, then the domain of the sender's email is returned.

Returns
the POP3 URL

Return type
str

`get_smtp_port()`

Get the port number used to establish an SMTP session.

Returns
the SMTP port

Return type
int

`get_smtp_url()`

Get the sender's SMTP URL to authenticate. If no SMTP URL was specified, then the domain of the sender's email is returned.

Returns
the SMTP URL

Return type
str

`set_email(email)`

Set the email address of the sender.

Parameters
`email` – the sender's email

Type
str

Raises
IOError - Invalid sender email

`set_password(password)`

Set the password of the sender email account to authenticate with the SMTP server.

Parameters
password – the sender's email password

Type
str

`set_pop3_port`(*pop3_port*)

Set the port number used to establish a POP3 session.

Parameters
pop3_port – the POP3 port

Type
int

`set_pop3_url`(*pop3_url*)

Set the sender's POP3 URL to authenticate.

Parameters
pop3_url – the POP3 URL

Type
str

`set_smtp_port`(*smtp_port*)

Set the port number used to establish an SMTP session.

Parameters
smtp_port – the SMTP port

Type
int

`set_smtp_url`(*smtp_url*)

Set the sender's SMTP URL to authenticate.

Parameters
smtp_url – the SMTP URL

Type
str

5.12 Parser Module

This module provides the necessary means to parse the sender, recipient and the email templates.

`heimdallsword.utils.parser.get_message`(*content_dir*, *recipient*)

Constructs a *heimdallsword.models.message.Message* object based on a given *heimdallsword.models.recipient.Recipient* and the path of the message file.

Parameters

- **content_dir** – the directory path where the message file resides
- **recipient** – the recipient assigned to the message

Type
str

Type
heimdallsword.models.recipient.Recipient

Returns
a *heimdallsword.models.message.Message* object

Return type
heimdallsword.models.message.Message

Raises
IOError - No recipient was provided

Raises
FileNotFoundError - Message file '{message_file_path}' was not found

Raises
ValueError - Failed to parse '{recipient.get_message_filename()}. No subject was provided

Raises
ValueError - Failed to parse '{recipient.get_message_filename()}'. The content type defined '{content_type}' is invalid. Content type can only be one of the following: {Message.PLAIN}, {Message.HTML}

Raises
KeyError - Failed to parse '{recipient.get_message_filename()}'. The tag {e} is not defined

Raises
ValueError - Failed to parse '{recipient.get_message_filename()}'. {e}. Use $$ to add the $ symbol

heimdallsword.utils.parser.**get_recipients**(*content_dir*, *recipients_file*)

Create a list of *heimdallsword.models.recipient.Recipient* objects based on the a given recipient file and parses the message file associated with each recipient.

Each line should be constructed in the following format:

email address, message text file

For example:

recipient1@example.com, msg1.txt

recipient2@example.com, msg2.txt

Key-values pairs can be appended after the message file and must be comma separated. Any key-value pairs appended requires that the key be in the message template, otherwise it will be ignored.

For example:

recipient1@example.com, msg1.txt, fname=John, lname=Smith

recipient2@example.com, msg2.txt, date=04/25/2022, transaction-id=ID10T

Using the first example, the key **fname** must be in the body of the message template file as such: ${fname}.

Parameters

- **content_dir** – the directory path where the message file resides
- **recipients_file** – the file containing the recipients

Type
str

Type
str

Returns
a list of *heimdallsword.models.recipient.Recipient* objects

Return type
heimdallsword.models.recipient.Recipient

Raises
NotADirectoryError - The content directory '{content_dir}' was not found

Raises
FileNotFoundError - Recipients file '{recipients_file}' was not found

Raises
ValueError - Failed to parse '{recipients_file}'. The '{recipients_file}' is empty

Raises
ValueError - Failed to parse '{recipients_file}'. Line #{line_counter} contains an invalid email address

Raises
FileNotFoundError - Failed to parse '{recipients_file}'. Line #{line_counter} does not contain a valid message file '{message_file}'

Raises
ValueError - Failed to parse '{recipients_file}'. Line #{line_counter} has an invalid key-value pair '{kv_pair.strip()}'

Raises
ValueError - Failed to parse '{recipients_file}'. Line #{line_counter} is not in the correct format

Raises
ValueError - Failed to parse '{recipients_file}'. No emails were found

heimdallsword.utils.parser.**get_senders**(*senders_file*, *default_smtp_port*, *default_pop3_port*)

Create a list of *heimdallsword.models.sender.Sender* objects based on the a given sender file.

Default ports for SMTP and POP3 must be provided which will be applied to all senders, unless each sender provides it's own SMTP port and POP3 port.

Each line should be constructed in the following format:

email address, password, smtp_url=, smtp_port=, pop3_url, pop3_port=

For example:

sender1@example.com, secretpassword!

sender2@example.com, P@$$w0rd, smtp_url=smtp.example.com, pop3_url=pop.example.com

sender3@example.com, hackyhackhack, smtp_url=smtp.example.com, smtp_port=587, pop3_url=pop.example.com, pop3_port=995

Parameters

- **senders_file** – the file containing one or multiple sender accounts
- **default_smtp_port** – the SMTP port to use for authentication
- **default_pop3_port** – the POP3 port to authenticate and read emails

Type
str

Type
int

Type
int

Raises
FileNotFoundError - Senders file '{senders_file}' was not found

Raises
ValueError - Failed to parse '{senders_file}'. The '{senders_file}' is empty

Raises
ValueError - Failed to parse '{senders_file}'. Line #{line_counter} contains an invalid email address

Raises
ValueError - Failed to parse '{senders_file}'. Line #{line_counter} has an invalid key-value pair '{kv_pair.strip()}'

Raises
ValueError - Failed to parse '{senders_file}'. Line #{line_counter} has an invalid SMTP port number

Raises
ValueError - Failed to parse '{senders_file}'. Line #{line_counter} has an invalid POP3 port number

Raises
ValueError - Failed to parse '{senders_file}'. Line #{line_counter} is not in the correct format

Raises
ValueError - Failed to parse '{senders_file}'. No emails were found

5.13 Util Module

This module contains general purpose functionalities that are used across the entire project.

`heimdallsword.utils.util.`**`calculate_elapsed_time`**(*start_time*, *stop_time*, *interval='default'*)

Calculates the time elapsed between two timestamps and returns the duration based on the type of interval specified.

The following *interval* types are supported:

- years
- days
- hours
- minutes
- seconds

This method is a modified version of a stackoverflow post by **Attaque** found at https://stackoverflow.com/a/47207182

Parameters

- **`start_time`** – the start time
- **`stop_time`** – the stop time (this must be greater than the start time)

Type
datetime

Type
datetime

Returns

- str - the total time elapsed between two timestamps
- int - the value based on the interval given

Raises

- IOError - start_time is not of datetime type
- IOError - stop_time is not of datetime type
- IOError - The interval provided is not supported: '{interval}'

heimdallsword.utils.util.**get_max_thread_count**()

Calculates the number of maximum threads based on the number of processors on the machine and multiply by 5. This is the same method used in the `concurrent.futures.ThreadPoolExecutor` module when *max_workers* is set to None.

Returns
max thread count

Return type
int

heimdallsword.utils.util.**get_value_from_key**(*key*, *data*)

Retrieves the value of a key-value pair given a key and the data in which the pair may exist.

Parameters

- **key** – the key used to identify the key-value pair
- **data** – the data in which the key-value pair may exist

Type
string

Type
string

Returns
the value of the key-value pair, otherwise an empty string

Return type
str

heimdallsword.utils.util.**is_directory_valid**(*arg_parser*, *dir_path*)

Callback method for validating the existance of a directory provided by the user as an argument. If the directory doesn't exist, it provides an error message to the `argparse.ArgumentParser` object.

Parameters
arg_parser – the argparser object that is parsing the directory argument

Type
`argparse.ArgumentParser`

Returns
True if the directory exists, False otherwise

Return type
bool

heimdallsword.utils.util.**is_email_valid**(*email*)

Checks to see if a given string follows the proper email format by using regular expressions.

The regular expression used in this method was based from:

- **URL:** https://stackoverflow.com/a/201378
- **Author:** bortzmeyer

However, **bortzmeyer**'s regular expression is a modified version of the original one found at http://emailregex.com. It was modified because according to **bortzmeyer**:

> "One RFC 5322 compliant regex can be found at the top of the page
> at http://emailregex.com/ but uses the IP address pattern that is
> floating around the internet with a bug that allows 00 for any of
> the unsigned byte decimal values in a dot-delimited address,
> which is illegal."

An additional modification was made to **bortzmeyer**'s modified version which validates uppercase letters in emails.

Parameters
email – the email string to validate

Type
string

Returns
True if email is in valid format, False otherwise

Return type
bool

heimdallsword.utils.util.**is_file_valid**(*arg_parser*, *file*)

Callback method for validating the existance of a file provided by the user as an argument. If the file doesn't exist, it provides an error message to the `argparse.ArgumentParser` object.

Parameters
arg_parser – the argparser object that is parsing the file argument

Type
`argparse.ArgumentParser`

Returns
True if the file exists, False otherwise

Return type
bool

heimdallsword.utils.util.**is_int**(*number*)

Determine if a given object is a whole number.

Returns
True if is a whole number, False otherwise

Return type
bool

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

H

I

L

M

N

O

P

R

S

T

U

W

Y