Honeycomb Documentation

Release 0.0.11

Cymmetria

May 15, 2018

Contents

Honeycomb is an open-source honeypot framework created by Cymmetria.

Honeycomb allows running honeypots with various integrations from a public library of plugins from https://github.com/Cymmetria/honeycomb_plugins

Writing new honeypot services and integrations for honeycomb is super easy! See the plugins repo for more info.

Full CLI documentation can be found at http://honeycomb.cymmetria.com/en/latest/cli.html

CHAPTER 1

Usage

Using pip:

```
$ pip install honeycomb-framework
$ honeycomb --help
```

Using Docker:

```
$ docker run -v honeycomb.yml:/usr/share/honeycomb/honeycomb.yml cymmetria/honeycomb
```

1.1 Running Honeycomb from command line

1.1.1 Honeycomb

Honeycomb is a honeypot framework.

```
Honeycomb [OPTIONS] COMMAND [ARGS]...
```

Options

-H, --home <home>
Honeycomb home path [default: /home/docs/.config/honeycomb]

--iamroot
Force run as root (NOT RECOMMENDED!)

-c, --config <config>
Path to a honeycomb.yml file that provides instructions

-v, --verbose
Enable verbose logging

--version
Show the version and exit.

Environment variables

DEBUG
Provide a default for *--verbose*

integration

Honeycomb integration commands.

```
Honeycomb integration [OPTIONS] COMMAND [ARGS]...
```

configure

Configure an integration with default parameters.

You can still provide one-off integration arguments to *honeycomb.commands.service.run()* if required.

```
Honeycomb integration configure [OPTIONS] INTEGRATION [ARGS]...
```

Options

-e, --editable
Load integration directly from unspecified path without installing (mainly for dev)

-a, --show_args
Show available integration arguments

Arguments

INTEGRATION
Required argument

ARGS
Optional argument(s)

install

Install a honeycomb integration from the online library, local path or zipfile.

```
Honeycomb integration install [OPTIONS] [INTEGRATIONS]...
```

Arguments

INTEGRATIONS
Optional argument(s)

list

List integrations.

```
Honeycomb integration list [OPTIONS]
```

Options

-r, --remote
Include available integrations from online repository

show

Show detailed information about a package.

```
Honeycomb integration show [OPTIONS] INTEGRATION
```

Options

-r, --remote
Show information only from remote repository

Arguments

INTEGRATION
Required argument

test

Execute the integration's internal test method to verify it's working as intended.

```
Honeycomb integration test [OPTIONS] [INTEGRATIONS]...
```

Options

-e, --editable
Run integration directly from specified path (main for dev)

Arguments

INTEGRATIONS
Optional argument(s)

uninstall

Uninstall a integration.

```
Honeycomb integration uninstall [OPTIONS] [INTEGRATIONS]...
```

Options

-y, --yes
Don't ask for confirmation of uninstall deletions.

Arguments

INTEGRATIONS
Optional argument(s)

service

Honeycomb service commands.

```
Honeycomb service [OPTIONS] COMMAND [ARGS]...
```

install

Install a honeypot service from the online library, local path or zipfile.

```
Honeycomb service install [OPTIONS] [SERVICES]...
```

Arguments

SERVICES
Optional argument(s)

list

List services.

```
Honeycomb service list [OPTIONS]
```

Options

-r, --remote
Include available services from online repository

logs

Show logs of daemonized service.

```
Honeycomb service logs [OPTIONS] SERVICES...
```

Options

-n, --num <num>
Number of lines to read from end of file [default: 10]

-f, --follow
Follow log output

Arguments

SERVICES
Required argument(s)

run

Load and run a specific service.

```
Honeycomb service run [OPTIONS] SERVICE [ARGS]...
```

Options

-d, --daemon
Run service in daemon mode

-e, --editable
Load service directly from specified path without installing (mainly for dev)

-a, --show-args
Show available service arguments

-i, --integration <integration>
Enable an integration

Arguments

SERVICE
Required argument

ARGS
Optional argument(s)

show

Show detailed information about a package.

```
Honeycomb service show [OPTIONS] SERVICE
```

Options

-r, --remote
Show information only from remote repository

Arguments

SERVICE
Required argument

status

Show status of installed service(s).

```
Honeycomb service status [OPTIONS] [SERVICES]...
```

Options

-a, --show-all
Show status for all services

Arguments

SERVICES
Optional argument(s)

stop

Stop a running service daemon.

```
Honeycomb service stop [OPTIONS] SERVICE
```

Options

-e, --editable
Load service directly from specified path without installing (mainly for dev)

Arguments

SERVICE
Required argument

test

Execute the service's internal test method to verify it's working as intended.

If there's no such method, honeycomb will attempt to connect to the port listed in config.json

```
Honeycomb service test [OPTIONS] [SERVICES]...
```

Options

-f, --force
Do not check if service is running before testing

-e, --editable
Run service directly from specified path (main for dev)

Arguments

SERVICES
Optional argument(s)

uninstall

Uninstall a service.

```
Honeycomb service uninstall [OPTIONS] [SERVICES]...
```

Options

-y, --yes
Don't ask for confirmation of uninstall deletions.

Arguments

SERVICES
Optional argument(s)

1.2 Running honeycomb in a container

The rationale of container support is to allow rapid configuration and deployment so launching honeypots would be simple and easy.

Since honeycomb is a standalone runner for services and integrations, it doesn't make sense for it to orchestrate deployment of external honeypots using docker. Instead, honeycomb itself could be run as a container.

This means the goal is to allow simple configuration that can be passed on to honeycomb and launch services with integration at ease.

To launch a honeycomb service with configured integration, the user needs to type in several commands to install a service, install an integration, configure that integration and finally run the service with optional parameters.

This actually resembles configuring a docker environment, where the user needs to type in several commands to define volumes, networks, and finally run a the desired container.

A yml configuration that specifies all of the desired configurations (services, integrations, etc.) will be supplied to honeycomb, and it will work like a state-machine to reach the desired state before finally running the service.

An example honeycomb file can be found on github

```
---
version: 1

services:
  simple_http:
    parameters:
      port: 1234

integrations:
  syslog:
    parameters:
      address: "127.0.0.1"
      port: 5514
      protocol: tcp
```

1.3 API Reference

1.3.1 honeycomb package

Subpackages

honeycomb.commands.service package

Submodules

honeycomb.commands.service.install module

Honeycomb service install command.

honeycomb.commands.service.list module

Honeycomb service list command.

honeycomb.commands.service.logs module

Honeycomb service logs command.

honeycomb.commands.service.run module

Honeycomb service run command.

honeycomb.commands.service.show module

Honeycomb service show command.

honeycomb.commands.service.status module

Honeycomb service status command.

honeycomb.commands.service.stop module

Honeycomb service stop command.

honeycomb.commands.service.test module

Honeycomb service test command.

honeycomb.commands.service.uninstall module

Honeycomb service uninstall command.

honeycomb.commands.integration package

Submodules

honeycomb.commands.integration.configure module

Honeycomb integration run command.

honeycomb.commands.integration.install module

Honeycomb integration install command.

honeycomb.commands.integration.list module

Honeycomb integration list command.

honeycomb.commands.integration.show module

Honeycomb integration show command.

honeycomb.commands.integration.test module

Honeycomb integration test command.

honeycomb.commands.integration.uninstall module

Honeycomb integration uninstall command.

honeycomb.decoymanager package

Submodules

honeycomb.decoymanager.models module

Honeycomb defs and constants.

class honeycomb.decoymanager.models.**Alert**(*alert_type: honeycomb.decoymanager.models.AlertType*, *id: str = NOTHING*, *status: int = 2*, *timestamp: datetime.datetime = NOTHING*) → None

Bases: object

Alert object.

ALERT_STATUS = ((0, 'Ignore'), (1, 'Mute'), (2, 'Alert'))

STATUS_ALERT = 2

STATUS_IGNORED = 0

STATUS_MUTED = 1

additional_fields

address

alert_type

cmd

decoy_hostname

decoy_ipv4

decoy_name

decoy_os

dest_ip

dest_port

domain

end_timestamp

event_description

event_type

file_accessed

id

image_file

image_md5

image_path

image_sha256

manufacturer

originating_hostname

originating_ip

originating_mac_address

originating_port

password

pid

ppid

request

status

timestamp

transport_protocol

uid

username

class honeycomb.decoymanager.models.**AlertType**(*name: str*, *label: str*, *service_type: honeycomb.servicemanager.models.ServiceType*) → None

Bases: object

Alert Type.

label

name

service_type

Module contents

Honeycomb Decoy Manager.

honeycomb.integrationmanager package

Submodules

honeycomb.integrationmanager.defs module

Honeycomb integrations definitions and constants.

class honeycomb.integrationmanager.defs.**IntegrationAlertStatuses** → None
Bases: *honeycomb.defs.IBaseType*

Provides information about the alert status in queue.

DONE = BaseNameLabel(name='done', label='Done')

ERROR_MISSING_SEND_FIELDS = BaseNameLabel(name='error_missing', label='Error. Missing

ERROR_POLLING = BaseNameLabel(name='error_polling', label='Error polling')

ERROR_POLLING_FORMATTING = BaseNameLabel(name='error_polling_formatting', label='Error

ERROR_SENDING = BaseNameLabel(name='error_sending', label='Error sending')

ERROR_SENDING_FORMATTING = BaseNameLabel(name='error_sending_formatting', label='Error

IN_POLLING = BaseNameLabel(name='in_polling', label='Polling')

PENDING = BaseNameLabel(name='pending', label='Pending')

POLLING = BaseNameLabel(name='polling', label='Polling')

class honeycomb.integrationmanager.defs.**IntegrationTypes** → None
Bases: *honeycomb.defs.IBaseType*

Integration types.

Currently only output event is supported.

EVENT_OUTPUT = BaseNameLabel(name='event_output', label='Event output')

honeycomb.integrationmanager.error_messages module

Honeycomb integration error messages.

honeycomb.integrationmanager.exceptions module

Honeycomb Output Integration Exceptions.

exception honeycomb.integrationmanager.exceptions.**IntegrationMissingRequiredFieldError**(**args*, ***kw*)

Bases: *honeycomb.exceptions.PluginError*

IntegrationMissingRequiredFieldError.

exception honeycomb.integrationmanager.exceptions.**IntegrationNoMethodImplementationError**(**a*, ***

Bases: *honeycomb.exceptions.PluginError*

IntegrationNoMethodImplementationError.

exception honeycomb.integrationmanager.exceptions.**IntegrationNotFound**(**args*, ***kwargs*)

Bases: *honeycomb.exceptions.PluginError*

Integration not found.

msg_format = 'Cannot find integration named {}, try installing it?'

exception honeycomb.integrationmanager.exceptions.**IntegrationOutputFormatError**(**args*, ***kwargs*)

Bases: *honeycomb.exceptions.PluginError*

IntegrationOutputFormatError.

exception honeycomb.integrationmanager.exceptions.**IntegrationPackageError**(**args*, ***kwargs*)

Bases: *honeycomb.exceptions.PluginError*

IntegrationPackageError.

exception honeycomb.integrationmanager.exceptions.**IntegrationPollEventError**(**args*, ***kwargs*)

Bases: *honeycomb.exceptions.PluginError*

IntegrationPollEventError.

exception honeycomb.integrationmanager.exceptions.**IntegrationSendEventError**(**args*, ***kwargs*)

Bases: *honeycomb.exceptions.PluginError*

IntegrationSendEventError.

msg_format = 'Error sending integration event: {}'

exception honeycomb.integrationmanager.exceptions.**IntegrationTestFailed**(**args*, ***kwargs*)

Bases: *honeycomb.exceptions.PluginError*

Integration not found.

msg_format = 'Integration test failed, details: {}'

honeycomb.integrationmanager.integration_utils module

Honeycomb Integration Manager.

class honeycomb.integrationmanager.integration_utils.**BaseIntegration**(*integration_data*)

Bases: `object`

Base Output Integration Class.

Will be overridden by output plugins.

format_output_data(*output_data*)
format_output_data.

poll_for_updates(*integration_output_data*)
poll_for_updates.

send_event(*required_alert_fields*)
Send event.

test_connection(*data*)
test_connection.

honeycomb.integrationmanager.models module

Honeycomb integration models.

class honeycomb.integrationmanager.models.**ConfiguredIntegration**(*name: str*, *path: str*, *integration: honeycomb.integrationmanager.models.Integr*, *send_muted: bool = False*, *created_at: datetime.datetime = NOTHING*) → None

Bases: object

Configured integration model.

class honeycomb.integrationmanager.models.**Integration**(*parameters: str*, *display_name: str*, *required_fields: list*, *polling_enabled: bool*, *integration_type: str*, *max_send_retries: int*, *supported_event_types: list*, *test_connection_enabled: bool*, *module=None*, *description: str = None*, *polling_duration: datetime.timedelta = 0*) → None

Bases: object

Integration model.

class honeycomb.integrationmanager.models.**IntegrationAlert**(*alert: honeycomb.decoymanager.models.Alert*, *status: str*, *retries: int*, *configured_integration: honeycomb.integrationmanager.models.ConfiguredIn*) → None

Bases: object

Integration alert model.

honeycomb.integrationmanager.registration module

Honeycomb service manager.

honeycomb.integrationmanager.registration.**get_integration_module**(*integration_path*)

Add custom paths to sys and import integration module.

Parameters **integration_path** – Path to integration folder

honeycomb.integrationmanager.registration.**register_integration**(*package_folder*)
Register a honeycomb integration.

Parameters **package_folder** – Path to folder with integration to load

Returns Validated integration object

Return type honeycomb.utils.defs.Integration()

honeycomb.integrationmanager.tasks module

Honeycomb integration tasks.

honeycomb.integrationmanager.tasks.**configure_integration**(*path*)
Configure and enable an integration.

honeycomb.integrationmanager.tasks.**create_integration_alert_and_call_send**(*alert*, *configured_integration*)
Create an IntegrationAlert object and send it to Integration.

honeycomb.integrationmanager.tasks.**get_current_datetime_utc**()
Return a datetime object localized to UTC.

honeycomb.integrationmanager.tasks.**get_valid_configured_integrations**(*alert*)
Return a list of integrations for alert filtered by alert_type.

Returns A list of relevant integrations

honeycomb.integrationmanager.tasks.**poll_integration_alert_data**(*integration_alert*)
Poll for updates on waiting IntegrationAlerts.

honeycomb.integrationmanager.tasks.**poll_integration_information_for_waiting_integration_ale**
poll_integration_information_for_waiting_integration_alerts.

honeycomb.integrationmanager.tasks.**send_alert_to_configured_integration**(*integration_alert*)
Send IntegrationAlert to configured integration.

honeycomb.integrationmanager.tasks.**send_alert_to_subscribed_integrations**(*alert*)
Send Alert to relevant integrations.

Module contents

Honeycomb Output Manager.

honeycomb.servicemanager package

Submodules

honeycomb.servicemanager.base_service module

Custom Service implementation from MazeRunner.

class honeycomb.servicemanager.base_service.**ServerCustomService**(*alert_types: list*, *service_args: dict = {}*) → None

Bases: multiprocessing.context.Process

Custom Service Class.

This class provides a basic wrapper for honeycomb and mazerunner services.

Parameters **service_args** – Validated dictionary of service arguments (see: honeycomb.Honeycomb.parse_service_args())

add_alert_to_queue(*alert_dict*)
Log alert and send to integrations.

emit(***kwargs*)
Send alerts to logfile.

Parameters **kwargs** – Fields to pass to *honeycomb.decoymanager.models.Alert*

logger = <logging.Logger object>

on_server_shutdown()
Shutdown function of the server.

Override this and take care of gracefully shutting down you service (e.g., close files)

on_server_start()
Service run loop function.

The service manager will call this function in a new thread.

Note: Must call *signal_ready()* after finishing configuration

---

run()
Daemon entry point.

run_service()
Run the service and start an alert processing queue.

See also:

Use *on_server_start()* and *on_server_shutdown()* for starting and shutting down your service

signal_ready()
Signal the service manager this service is ready for incoming connections.

honeycomb.servicemanager.defs module

Honeycomb services definitions and constants.

honeycomb.servicemanager.defs.**ALLOWED_PROTOCOLS = ['TCP', 'UDP']**
Parameters.

honeycomb.servicemanager.defs.**STDERRLOG = 'stderr.log'**
Service section.

honeycomb.servicemanager.error_messages module

Honeycomb services error messages.

honeycomb.servicemanager.exceptions module

Honeycomb Service Manager Exceptions.

exception honeycomb.servicemanager.exceptions.**ServiceManagerException**(**args*, ***kwargs*)

Bases: *honeycomb.exceptions.PluginError*

Generic Service Manager Exception.

exception honeycomb.servicemanager.exceptions.**ServiceNotFound**(**args*, ***kwargs*)

Bases: *honeycomb.servicemanager.exceptions.ServiceManagerException*

Specified service does not exist.

msg_format = 'Cannot find service named {}, try installing it?'

exception honeycomb.servicemanager.exceptions.**UnsupportedOS**(**args*, ***kwargs*)

Bases: *honeycomb.servicemanager.exceptions.ServiceManagerException*

Specified service does not exist.

msg_format = 'Service requires running on {} and you are using {}'

honeycomb.servicemanager.models module

Honeycomb service models.

class honeycomb.servicemanager.models.**OSFamilies** → None

Bases: *honeycomb.defs.IBaseType*

Defines supported platforms for services.

ALL = BaseNameLabel(name='All', label='All')

LINUX = BaseNameLabel(name='Linux', label='Linux')

MACOS = BaseNameLabel(name='Darwin', label='Darwin')

WINDOWS = BaseNameLabel(name='Windows', label='Windows')

class honeycomb.servicemanager.models.**ServiceType**(*name: str*, *ports: list*, *label: str*, *allow_many: bool*, *supported_os_families: list*, *alert_types: list = []*) → None

Bases: object

Holds loaded service metadata.

honeycomb.servicemanager.registration module

Honeycomb service manager.

honeycomb.servicemanager.registration.**get_service_module**(*service_path*)

Add custom paths to sys and import service module.

Parameters **service_path** – Path to service folder

honeycomb.servicemanager.registration.**register_service**(*package_folder*)
Register a honeycomb service.

Parameters **package_folder** – Path to folder with service to load

Returns Validated service object

Return type honeycomb.utils.defs.ServiceType()

Module contents

Honeycomb Service Manager.

honeycomb.utils package

Submodules

honeycomb.utils.config_utils module

Honeycomb Config Utilities.

honeycomb.utils.config_utils.**config_field_type**(*field*, *cls*)
Validate a config field against a type.

Similar functionality to *validate_field_matches_type()* but returns *honeycomb.defs.ConfigField*

honeycomb.utils.config_utils.**get_config_parameters**(*plugin_path*)
Return the parameters section from config.json.

honeycomb.utils.config_utils.**get_truetype**(*value*)
Convert a string to a pythonized parameter.

honeycomb.utils.config_utils.**is_valid_field_name**(*value*)
Ensure field name is valid.

honeycomb.utils.config_utils.**process_config**(*ctx*, *configfile*)
Process a yaml config with instructions.

This is a heavy method that loads lots of content, so we only run the imports if its called.

honeycomb.utils.config_utils.**validate_config**(*config_json*, *fields*)
Validate a JSON file configuration against list of *honeycomb.defs.ConfigField*.

honeycomb.utils.config_utils.**validate_config_parameters**(*config_json*, *allowed_keys*, *allowed_types*)
Validate parameters in config file.

honeycomb.utils.config_utils.**validate_field**(*field*, *allowed_keys*, *allowed_types*)
Validate field is allowed and valid.

honeycomb.utils.config_utils.**validate_field_matches_type**(*field*, *value*, *field_type*, *select_items=None*, *_min=None*, *_max=None*)
Validate a config field against a specific type.

honeycomb.utils.daemon module

Honeycomb DaemonRunner utility.

class honeycomb.utils.daemon.**myRunner**(*app*, *pidfile=None*, *stdout=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *stderr=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*, *stdin=<_io.TextIOWrapper name='/dev/null' mode='rt' encoding='UTF-8'>*)

Bases: daemon.runner.DaemonRunner

Overriding default runner behaviour to be simpler.

honeycomb.utils.plugin_utils module

Honeycomb generic plugin install utils.

exception honeycomb.utils.plugin_utils.**CTError**(*errors*)

Bases: Exception

Copytree exception class, used to collect errors from the recursive copy_tree function.

honeycomb.utils.plugin_utils.**copy_file**(*src*, *dst*)

Copy a single file.

:param:src: Source name :param:dst: Destination name

honeycomb.utils.plugin_utils.**copy_tree**(*src*, *dst*, *symlinks=False*, *ignore=[]*)

Copy a full directory structure.

:param:src: Source path :param:dst: Destination path :param:symlinks: Copy symlinks :param:ignore: Subdirs/filenames to ignore

honeycomb.utils.plugin_utils.**get_plugin_path**(*home*, *plugin_type*, *plugin_name*, *editable=False*)

Return path to plugin.

:param:home: Path to honeycomb home :param:plugin_type: Type of plugin (honeycomb.defs.SERVICES pr honeycomb.defs.INTEGRATIONS) :param:plugin_name: Name of plugin :param:editable: Use plugin_name as direct path instead of loading from honeycomb home folder

honeycomb.utils.plugin_utils.**get_select_items**(*items*)

Return list of possible select items.

honeycomb.utils.plugin_utils.**install_deps**(*pkgpath*)

Install plugin dependencies using pip.

We import pip here to reduce load time for when its not needed.

honeycomb.utils.plugin_utils.**install_dir**(*pkgpath*, *install_path*, *register_func*, *delete_after_install=False*)

Install plugin from specified directory.

install_path and register_func are same as *install_plugin()*. :delete_after_install: Delete pkgpath after install (used in *install_from_zip()*).

honeycomb.utils.plugin_utils.**install_from_repo**(*pkgname*, *plugin_type*, *install_path*, *register_func*)

Install plugin from online repo.

honeycomb.utils.plugin_utils.**install_from_zip**(*pkgpath*, *install_path*, *register_func*, *delete_after_install=False*)

Install plugin from zipfile.

honeycomb.utils.plugin_utils.**install_plugin**(*pkgpath*, *plugin_type*, *install_path*, *register_func*)

Install specified plugin.

:param:pkgpath: Name of plugin to be downloaded from online repo or path to plugin folder or zip file. :param:install_path: Path where plugin will be installed. :param:register_func: Method used to register and validate plugin.

honeycomb.utils.plugin_utils.**list_local_plugins**(*plugin_type*, *plugins_path*, *plugin_details*)

List local plugins with details.

honeycomb.utils.plugin_utils.**list_remote_plugins**(*installed_plugins*, *plugin_type*)

List remote plugins from online repo.

honeycomb.utils.plugin_utils.**parse_plugin_args**(*command_args*, *config_args*)

Parse command line arguments based on the plugin's parameters config.

Parameters

- **command_args** – Command line arguments as provided by the user in *key*=*value* format.
- **config_args** – Plugin parameters parsed from config.json.

Returns Validated dictionary of parameters that will be passed to plugin class

honeycomb.utils.plugin_utils.**print_plugin_args**(*plugin_path*)

Print plugin parameters table.

honeycomb.utils.plugin_utils.**uninstall_plugin**(*pkgpath*, *force*)

Uninstall a plugin.

:param:pkgpath: Path to package to uninstall (delete) :param:force: Force uninstall without asking

honeycomb.utils.tailer module

Honeycomb service log tailer.

class honeycomb.utils.tailer.**Tailer**(*name: str*, *filepath: str*, *color: str = ''*, *nlines: int = 10*, *follow: bool = False*, *outfile=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *sleeptime: int = 0.5*, *show_name: bool = True*, *used_colors: list = []*) → None

Bases: object

Colorized file tailer.

Print lines from a file prefixed with a colored name. Optionally continue to follow file.

follow_file()

Follow a file and send every new line to a callback.

print_log(*line*)

Print a line from a logfile.

print_named_log(*line*)

Print a line from a logfile prefixed with service name.

stop()
Stop follow.

honeycomb.utils.validators module

Honeycomb generic validators.

honeycomb.utils.validators.**validate_ip_or_hostname**(*value*)
IP/Host parameter validator.

honeycomb.utils.validators.**validate_port**(*value*)
Validate port is in standard range.

honeycomb.utils.wait module

Honeycomb wait utilities.

exception honeycomb.utils.wait.**TimeoutException**
Bases: Exception

Exception to be raised on timeout.

honeycomb.utils.wait.**search_json_log**(*filepath*, *key*, *value*)
Search json log file for a key=value pair.

Parameters

- **filepath** – Valid path to a json file
- **key** – key to match
- **value** – value to match

Returns First matching line in json log file, parsed by json.loads()

honeycomb.utils.wait.**wait_until**(*func*, *check_return_value=True*, *total_timeout=60*, *interval=0.5*, *exc_list=None*, *error_message=''*, **args*, ***kwargs*)
Run a command in a loop until desired result or timeout occurs.

Parameters

- **func** – Function to call and wait for
- **check_return_value** (*bool*) – Examine return value
- **total_timeout** (*int*) – Wait timeout,
- **interval** (*float*) – Sleep interval between retries
- **exc_list** (*list*) – Acceptable exception list
- **error_message** (*str*) – Default error messages
- **args** – args to pass to func
- **kwargs** – lwargs to pass to fun

Module contents

Honeycomb Utils.

Submodules

honeycomb.cli module

Honeycomb Command Line Interface.

class honeycomb.cli.**MyLogger**(*name*, *level=0*)
Bases: logging.Logger

Custom Logger.

makeRecord(*name*, *level*, *fn*, *lno*, *msg*, *args*, *exc_info*, *func=None*, *extra=None*, *sinfo=None*)
Override default logger to allow overriding of internal attributes.

honeycomb.cli.**setup_logging**(*home*, *verbose*)
Configure logging for honeycomb.

honeycomb.defs module

Honeycomb defs and constants.

class honeycomb.defs.**BaseCollection** → None
Bases: object

Abstract type collection mixin, should hold BaseNameLabel attributes.

class honeycomb.defs.**BaseNameLabel**(*name*, *label*) → None
Bases: object

Generic name/label class.

honeycomb.defs.**CONFIG_FILE_NAME = 'config.json'**
Parameters constants.

class honeycomb.defs.**ConfigField**(*validator_func*, *get_error_message*) → None
Bases: object

Config Validator.

error_message is also a function to calculate the error when we ran the validator_func

honeycomb.defs.**GITHUB_RAW_URL = 'https://raw.githubusercontent.com/Cymmetria/honeycomb_plug**
Config constants.

class honeycomb.defs.**IBaseType** → None
Bases: object

Abstract type interface, provides BaseNameLabel collection methods.

classmethod all_labels()
Return list of all property labels.

classmethod all_names()
Return list of all property names.

honeycomb.error_messages module

Honeycomb generic error messages.

honeycomb.exceptions module

Honeycomb Exceptions.

exception honeycomb.exceptions.**BaseHoneycombException**(**args*, ***kwargs*)
Bases: click.exceptions.ClickException

Base Exception.

msg_format = None

exception honeycomb.exceptions.**ConfigFieldMissing**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.ConfigValidationError*

Field is missing from config file.

msg_format = 'field {} is missing from config file'

exception honeycomb.exceptions.**ConfigFieldTypeMismatch**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.ConfigValidationError*

Config field does not match specified type.

msg_format = 'Parameters: Bad value for {}={} (must be {})'

exception honeycomb.exceptions.**ConfigFieldValidationError**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.ConfigValidationError*

Error validating config field.

msg_format = 'Failed to import config. error in field {} with value {}: {}'

exception honeycomb.exceptions.**ConfigFileNotFound**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.PluginError*

Config file not found.

msg_format = 'Missing file {}'

exception honeycomb.exceptions.**ConfigValidationError**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.BaseHoneycombException*

Base config validation error.

exception honeycomb.exceptions.**ParametersFieldError**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.ConfigValidationError*

Error validating parameter.

msg_format = "Parameters: '{}' is not a valid {}"

exception honeycomb.exceptions.**PathNotFound**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.BaseHoneycombException*

Specified path was not found.

msg_format = 'Cannot find path {}'

exception honeycomb.exceptions.**PluginAlreadyInstalled**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.PluginError*

Plugin already installed.

msg_format = '{} is already installed'

exception honeycomb.exceptions.**PluginError**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.BaseHoneycombException*

Base Plugin Exception.

exception honeycomb.exceptions.**PluginNotFoundInOnlineRepo**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.PluginError*

Plugin not found in online repo.

msg_format = 'Cannot find {} in online repository'

exception honeycomb.exceptions.**PluginRepoConnectionError**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.PluginError*

Connection error when trying to connect to plugin repo.

msg_format = 'Unable to access online repository (check debug logs for detailed info)'

exception honeycomb.exceptions.**RequiredFieldMissing**(**args*, ***kwargs*)
Bases: *honeycomb.exceptions.PluginError*

Required parameter is missing.

msg_format = "Parameters: '{}' is missing (use --args to see all parameters)"

Python Module Index

h

Index

Symbols

A

B

I

L

M

N

O

P

R

S

T

U

V

W