cfg_load Documentation

Release 0.9.0

Martin Thoma

Aug 22, 2020

CONTENTS:

CHAPTER

ONE

CFG_LOAD

cfg_load.**load**(*filepath: str*, *load_raw: bool = False*, *load_remote: bool = True*, ***kwargs: Any*) → Union[*cfg_load.Configuration*, Dict]

Load a configuration file.

Parameters

- **filepath** (*str*) – Path to the configuration file.
- **load_raw** (*bool, optional (default: False)*) – Load only the raw configuration file as a dict, without applying any logic to it.
- **load_remote** (*bool, optional (default: True)*) – Load files stored remotely, e.g. from a webserver or S3
- ****kwargs** – Arbitrary keyword arguments which get passed to the loader functions.

Returns config

Return type *Configuration*

class cfg_load.**Configuration**(*cfg_dict: Dict*, *meta: Dict*, *load_remote: bool = True*)

Configuration class.

Essentially, this is an immutable dictionary.

Parameters

- **cfg_dict** (*Dict*) –
- **meta** (*Dict*) –
- **load_remote** (*bool*) –

apply_env(*env_mapping: List[Dict[str, Any]]*) → *cfg_load.Configuration*

Apply environment variables to overwrite the current Configuration.

The env_mapping has the following structure (in YAML):

``` - env_name: “AWS_REGION”

keys: [“AWS”, “REGION”] converter: str

- env_name: “AWS_IS_ENABLED” keys: [“AWS”, “IS_ENABLED”] converter: bool

```

If the env_name is not an ENVIRONMENT variable, then nothing is done. If an ENVIRONMENT variable is not defined in env_mapping, nothing is done.

Known converters:

- bool
- str
- str2str_or_none
- int
- float
- json

Parameters **env_mapping** (Configuration) –

Returns **update_config**

Return type *Configuration*

pformat (*indent: int = 4, meta: bool = False*) → str
Pretty-format the configuration.

Parameters

- **indent** (*int*) –
- **meta** (*bool*) – Print metadata

Returns **pretty_format_cfg**

Return type str

set (*key: str, value: Any*) → *cfg_load.Configuration*
Set a value in the configuration.

Although it is discouraged to do so, it might be necessary in some cases.

If you need to overwrite a dictionary, then you should do:

>> inner_dict = cfg['key'] >> inner_dict['inner_key'] = 'new_value' >> cfg.set('key', inner_dict)

to_dict () → Dict
Return a dictionary representation of the configuration.

This does NOT contain the metadata connected with the configuration. It is discuraged to use this in production as it loses the metadata and guarantees connected with the configuraiton object.

Returns **config**

Return type dict

update (*other:* cfg_load.Configuration) → *cfg_load.Configuration*
Update this configuration with values of the other configuration.

other : Configuration

Returns **updated_config**

Return type *Configuration*

CHAPTER

TWO

INDICES AND TABLES

- modindex
- search

INDEX