glorpen-config Documentation

Release 2.2.0

Arkadiusz Dzięgiel

Feb 03, 2020

Contents

Config framework for Your projects - with validation, interpolation and value normalization!

CHAPTER 1

Official repositories

GitHub: https://github.com/glorpen/glorpen-config

BitBucket: https://bitbucket.org/glorpen/glorpen-config

CHAPTER 2

Features

You can:

- create custom fields for custom data
- define configuration schema inside Python app
- convert configuration values to Python objects
- validate configuration
- use interpolation to fill config values
- set default values

2.1 Loading data

glorpen.config.Config uses *glorpen.config.loaders* to allow loading data from different sources.

Loaders should accept:

- path, `filepath` constructor argument
- file-like object, `fileobj` constructor argument

Additionally you can just pass `dict` data to config with *glorpen.config.Config.load_data()* or *glorpen.config.Config.finalize()*.

2.2 Interpolation

You can reuse values from config with `{{ path.to.value }}` notation, eg:

```
project:
   path: "/tmp"
   cache_path: "{{ project.path }}/cache"
```

String interpolation currently can be used only with `glorpen.config.fields.String` fields.

2.3 Normalization and validation

Each field type has own normalization rules, eg. for *glorpen.config.fields.log.LogLevel*:

```
logging: DEBUG
```

`config.get("logging")` would yield value `10` as is `logging.DEBUG`.

Additionally it will raise *glorpen.config.exceptions.ValidationError* if invalid level name is given.

2.4 Default values

Each field can have default value. If no value is given in config but default one is set, it will be used instead.

Default values adhere to same interpolation and normalization rules - each default value is denormalized and then passed to normalizers. That way complex object can still profit from config interpolation. There should not be any real impact on performance as it is done only once.

CHAPTER 3

Contents

3.1 Example usage

3.1.1 Using fields

Your first step should be defining configuration schema:

```
import logging
import glorpen.config.fields.simple as f
from glorpen.config.fields.log import LogLevel

project_path = "/tmp/project"

spec = f.Dict({
  "project_path": f.Path(default=project_path),
  "project_cache_path": f.Path(default="{{ project_path }}/cache"),
  "logging": fl.LogLevel(default=logging.INFO),
  "database": f.String(),
  "sources": f.Dict({
      "some_param": f.String(),
      "some_path": f.Path(),
  }),
  "maybe_string": f.Variant([
      f.String(),
      f.Number()
  ])
})
```

Example yaml config:

```
logging: "DEBUG"
database: "mysql://...."
sources:
  some_param: "some param"
```

(continues on next page)

(continued from previous page)

```
  some_path: "/tmp"
maybe_string: 12
```

Then you can create `glorpen.config.Config` instance:

```
from glorpen.config import Config
import glorpen.config.loaders as loaders

loader = loaders.YamlLoader(filepath=config_path)
cfg = Config(loader=loader, spec=spec).finalize()

cfg.get("sources.some_param") #=> 'some param'
cfg.get("project_path") #=> '/tmp/project'
cfg.get("project_cache_path") #=> '/tmp/project/cache'
cfg.get("logging") #=> 10
cfg.get("maybe_string") #=> 12
```

3.1.2 Creating custom fields

Custom field class should extend *glorpen.config.fields.base.Field* or *glorpen.config.fields.base.FieldWithDefault*.

glorpen.config.fields.base.Field.make_resolvable() method should register normalizer functions which later will be called in registration order. Each value returned by normalizer is passed to next one. After chain end value is returned as config value.

Returned *glorpen.config.fields.base.ResolvableObject* instance is resolved before passing it to next normalizer.

If value passed to normalizator is invalid it should raise *glorpen.config.exceptions.ValidationError*. Sometimes value can be lazy loaded - it is represented as *glorpen.config.fields.base.ResolvableObject*. You can get real value by using *glorpen.config.fields.base.resolve()*.

```
class MyValue(object):
   def __init__(self, value):
      super(MyValue, self).__init__()
      self.value = value

class MyField(Field):

    def to_my_value(self, value, config):
        return MyValue(value)

    def is_value_supported(self, value):
        return True

    def make_resolvable(self, r):
        r.on_resolve(self.to_my_value)
```

The last thing is to use prepared custom field in configuration spec.

3.2 glorpen.config API Documentation

3.2.1 glorpen.config

glorpen.config.**__version__**
Current package version.

class glorpen.config.**Config**(*spec*, *loader=None*, *split_character='.'*)
Config validator and normalizer.

finalize(*data=None*)
Load and resolve configuration in one go.

If data argument is given loader specified in constructor will not be used.

get(*p*)
Gets value from config. To get value under *some_key* use dotted notation: *some_key.value* (defaults).

load_data(*data*)
Loads given data as source.

resolve()
Visits all values and converts them to normalized form.

3.2.2 glorpen.config.fields.base

class glorpen.config.fields.base.**Field**
Single field in configuration file.

Custom fields should register own resolvers/validators/normalizers by extending *make_resolvable()*.

For handling registered callbacks, see *ResolvableObject*.

is_value_supported(*value*)
Checks if provided value is supported by this field

make_resolvable(*r*)
Used to register normalizes in current *ResolvableObject*.

resolve(*v*, *checked=False*)
Wraps value in *ResolvableObject* optionally checking whether provided value is supported.

class glorpen.config.fields.base.**FieldWithDefault**(*default=<class glorpen.config.fields.base._UnsetValue>*, *allow_blank=False*)
Base class for nullable fields with defaults.

class glorpen.config.fields.base.**ResolvableObject**(*o*)
Configuration value ready to be resolved.

Callbacks are registered by calling on_resolve.

To each callback are passed:

- currently handled value
- *Config* instance

By using *Config* instance you can realize value based on any other value in configuration.

If value is invalid, callback should raise *ValidationError* with appropriate error message.

on_resolve(*f*)
Registers given callback to run when resolving values. Passed function should accept following arguments:

- value - may be a value or *ResolvableObject*
- a *Config* instance

resolve(*config*)
Resolves value with given config

glorpen.config.fields.base.**path_validation_error**(**args*, ***kwds*)
Adds given path to validation error to make exceptions easy to read

glorpen.config.fields.base.**resolve**(*obj*, *config*)
Returns real object value (resolved) if applicable

3.2.3 glorpen.config.fields.simple

class glorpen.config.fields.simple.**Any**(*default=<class glorpen.config.fields.base._UnsetValue>*, *allow_blank=False*)
Field that accepts any value.

class glorpen.config.fields.simple.**Dict**(*schema=None*, *keys=None*, *values=None*, ***kwargs*)
Converts values to collections.OrderedDict

Supports setting whole schema (specific keys and specific values) or just keys type and values type.

Dict values are lazy resolved.

__init__(*schema=None*, *keys=None*, *values=None*, ***kwargs*)
To set specific schema pass dict to schema argument: `{"param1": SomeField()}`.

To specify keys and values type use keys and values arguments: `Dict(keys=String(), values=Number())`.

class glorpen.config.fields.simple.**List**(*schema*, ***kwargs*)
Converts value to list.

List values are lazy resolved.

class glorpen.config.fields.simple.**Number**(*default=<class glorpen.config.fields.base._UnsetValue>*, *allow_blank=False*)
Converts value to numbers.

class glorpen.config.fields.simple.**Path**(*default=<class glorpen.config.fields.base._UnsetValue>*, *allow_blank=False*)
Converts given value to disk path.

class glorpen.config.fields.simple.**PathObj**(*default=<class glorpen.config.fields.base._UnsetValue>*, *allow_blank=False*)
Converts value to pathlib.Path object

class glorpen.config.fields.simple.**String**(*default=<class glorpen.config.fields.base._UnsetValue>*, *allow_blank=False*)
Converts value to string with optional interpolation.

class glorpen.config.fields.simple.**Variant**(*schema*, *try_resolving=False*)
Converts value to normalized state using one *Field* chosen from multiple provided.

To allow blank values you have to pass child field with enabled blank values. First field which supports value (*Field.is_value_supported()*) will be used to convert it.

When `try_resolving` mode is disabled (default), value for child fields will only be checked with `is_value_supported`, so resulting field will be based only of data type, not value.

When enabled, in addition to checking for supported values data will be resolved and first non error result used.

3.2.4 glorpen.config.fields.log

class glorpen.config.fields.log.**LogLevel**(*default=<class glorpen.config.fields.base._UnsetValue>*, *allow_blank=False*)
Converts log level name to internal number for use with `logging`

3.2.5 glorpen.config.fields.version

3.2.6 glorpen.config.loaders

class glorpen.config.loaders.**BaseLoader**(*filepath=None*, *fileobj=None*)
Base class for any loader.

_parse(*data*)
Extending classes should overwrite this method with parsing logic.

_setup()
Extending classes can use it to setup loader.

load()
Reads source specified in constructor.

class glorpen.config.loaders.**YamlLoader**(*filepath=None*, *fileobj=None*)
Reads yaml files.

3.2.7 glorpen.config.exceptions

exception glorpen.config.exceptions.**CircularDependency**(**args*, ***kwargs*)
Thrown when interpolation causes loop.

exception glorpen.config.exceptions.**ConfigException**
Base exception for config errors.

exception glorpen.config.exceptions.**PathValidationError**(*validation_error*)
Exception for improved readability - uses *ValidationError* to provide full path to field with error.

exception glorpen.config.exceptions.**ValidationError**(*message*, **args*)
Exception for when there is error in validation of values in fields.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

Symbols

A

B

C

D

F

G

I

L

M

N

O

P

R

S

V

Y