wheezy.validation documentation

Release latest

Andriy Kornatskyy

Apr 17, 2021

Contents

CHAPTER 1

Introduction

wheezy.validation is a python package written in pure Python code. It is a lightweight validation library.

It is optimized for performance, well tested and documented.

Resources:

- source code and issues tracker are available on github
- documentation

CHAPTER 2

Contents

2.1 Getting Started

2.1.1 Install

wheezy.validation requires python version 3.6+. It is independent of operating system. You can install it from pypi site:

```
$ pip install wheezy.validation
```

2.2 Examples

We start with a simple credential validation example. Before we proceed let setup virtualenv environment:

```
$ pip install wheezy.validation
```

2.2.1 Credential Validation

Domain Model

Our domain model requires that user enter valid credentials during sigin, so `Credential` model might look like this:

```
class Credential(object):

    def __init__(self):
        self.username = ''
        self.password = ''
```

Validator

We know that username is a string between 2 and 20 in length, while password must have at least 8 characters and we would like limit it by 12. Both username and password are required and we need a separate message for this case. Here is `credential_validator` that serves our purpose:

```
from wheezy.validation import Validator
from wheezy.validation.rules import length
from wheezy.validation.rules import required


credential_validator = Validator({
    'username': [required, length(min=2, max=20)],
    'password': [required, length(min=8, max=12)]
})
```

Since validator in no way dependent on object it is going to validate, it can be reused in any combination, the only requirement that attributes defined in validator must exist in object you are going to validate.

Validation

Now we can proceed with validation:

```
credential = Credential()
errors = {}
succeed = credential_validator.validate(credential, results=errors)
```

`errors` dictionary contains all errors reported during validation. Key corresponds to attribute being checked, while value is a list of errors.

2.2.2 Credential Update

Web form submit is a dictionary where key is the name of the input element being submitted and value is a list. That list can have just one item for elements like input or several values that depict user choice.

Let assume form submitted values look this way:

```
values = {'username': [''], 'password': ['']}
```

Our attempt to update `Credential` model with `values`:

```
from wheezy.validation.model import try_update_model

credential = Credential()
errors = {}
succeed = try_update_model(credential, values, errors)
```

`errors` dictionary contains all errors reported during model update. Key corresponds to attribute being updated, while value is a list of errors.

2.3 User Guide

Users fill the web form with necessary information, however, they often make mistakes. This is where form validation comes into play. The goal of validation is to ensure that the user provided necessary and properly formatted information

needed to successfully complete an operation.

2.3.1 Validator

Validator is a container of validation rules that all together provide object validation. You instantiate *Validator* and supply it a map between attribute names being validated and list of rules. Here is an example:

```
credential_validator = Validator({
    'username': [required, length(max=10)],
    'password': [required, length(min=8, max=12)]
})
```

Validator in no way tied to object and/or class being validated, instead the only requirement is existance of attributes being validated.

Validator supports __getitem__ interface, so is applicable to `dict` like objects:

```
user = {'username': '', 'password': ''}
errors = {}
succeed = credential_validator.validate(user, errors)
```

Method `validate` returns `True` only in case all validation rules succeed otherwise `False`.

`errors` dictionary contains all errors reported during validation. Key corresponds to attribute name being checked, while value is a list of errors.

If you need validation check all rules for failed attribute, you need set `stop` attribute to `False` (default is to stop on first error):

```
succeed = credential_validator.validate(user, errors, stop=False)
```

Nested Validator

Validator can be nested into some other validator, so ultimately can form any hierarchically complex structure. This can be useful for composite objects, e.g. `Registration` model can aggregate `Credential` model. While each model has own validation, registration model can nest the validator for the credential model:

```
class Registration(object):
    def __init__(self):
        self.credential = Credential()

registration_validator = Validator({
    'credential': credential_validator
})
```

Internationalization

Validator supports the python standard `gettext` module. You need to pass `gettext` translations as a argument to `validate` method. Here is an example:

```
from gettext import NullTranslations

translations = NullTranslations()
succeed = credential_validator.validate(
```

(continues on next page)

(continued from previous page)

```
        user,
        errors,
        translations=translations)
```

Thread Safety

Validator does not alter its state once initialized. It is guaranteed to be thread safe.

2.3.2 Validation Rules

Validation rules prevent bad data from being processed. A validation rule is a criterion used in the process of data validation. Rules support simple types attributes as well as list typ attributes, e.g. `iterator` rule can apply a number of other rules to each item in the list.

There are a number of validation rules already defined.

- `required`. Any value evaluated to boolean `True` passes this rule. Also take a look at the `required_but_missing` list. See *RequiredRule*.
- `not_none`. None value will not pass this rule. See *NotNoneRule*.
- `missing`, `empty`. Any value evaluated to boolean `False` passes this rule. Also take a look at the `required_but_missing` list. See *RequiredRule*.
- `length`. Result of python function `len()` must fall within a range defined by this rule. Supported range attributes include: `min`, `max`. See *LengthRule*.
- `compare`. Compares attribute being validated with some other attribute value. Supported comparison operations include: `equal`, `not_equal`. See *CompareRule*.
- `predicate`, `model_predicate`. Fails if predicate returns boolean `False`. Predicate is any callable that accepts a model and returns a boolean. It is useful for custom rules, e.g. a number of days between two model properties must not exceed a certain value, etc. See *PredicateRule*.
- `must`, `value_predicate`. Fails if predicate returns boolean `False`. Predicate is any callable that accepts a value and returns a boolean. It is useful for custom rule applicable to multiple attributes of model. See *ValuePredicateRule*.
- `regex`. Search for regular expression pattern. Initialized with `regex` as a regular expression pattern or a pre-compiled regular expression. Supports `negated` argument. See *RegexRule*.
- `slug`. Ensures only letters, numbers, underscores or hyphens. See *SlugRule*.
- `email`. Ensures a valid email. See *EmailRule*.
- `scientific`. Ensures a valid scientific string input. See *ScientificRule*.
- `base64`. Ensures a valid base64 string input (supports alternative alphabet for `+` and `/` characters). See *Base64Rule*.
- `urlsafe_base64`. Ensures a valid base64 string input using an alphabet, which substitutes `-` instead of `+` and `_` instead of `/` in the standard Base64 alphabet. The input can still contain `=`. See *URLSafeBase64Rule*.
- `range`. Ensures value is in range defined by this rule. Works with any numbers including int, float, decimal, etc. Supported range attributes include: `min`, `max`. See *RangeRule*.
- `and_`. Applies all `rules` regardless of validation result. See *AndRule*.

- `or_`. Succeeds if at least one rule in `rules` succeed. Failed rule results are not added unless they all fail. See *OrRule*.
- `iterator`. Applies `rules` to each item in value. Iterates over each rule and checks whenever any item in value fails. Designed to work with iteratable attributes: list, tuple, etc. See *IteratorRule*.
- `one_of`. Value must match at least one element from `items`. Checks whenever value belongs to `items`. See *OneOfRule*.
- `relative_date`, `relative_utcdate`, `relative_tzdate`, `relative_datetime`, `relative_utcdatetime`, `relative_tzdatetime`. Check if value is in relative date/datetime range per local, UTC or tz time. See *RelativeDateDeltaRule*, *RelativeUTCDateDeltaRule*, *RelativeTZDateDeltaRule* and *RelativeDateTimeDeltaRule*, *RelativeUTCDateTimeDeltaRule*, *RelativeTZDateTimeDeltaRule*.
- `relative_timestamp`, `relative_unixtime`. Check if value is in relative unix timestamp range. See *RelativeUnixTimeDeltaRule*.
- `adapter`, `int_adapter`. Adapts a value according to converter. This is useful when you need to keep string input in model but validate as an integer. See *AdapterRule*, *IntAdapterRule*.
- `ignore`. The idea behind this rule is to be able to substitute any validation rule by this one that always succeeds. See *IgnoreRule*.

Custom Message

You are able to customize the error message by using `message_template` argument during rule declaration:

```
credential_validator = Validator({
    'username': [required(message_template='Required field')]
})
```

Every rule supports `message_template` argument during rule declaration.

`gettext` utilities

Please remember to add `msgid`/`msgstr` of customized validation error to `po` file. You can extract gettext messages by:

```
$ xgettext --join-existing --sort-by-file --omit-header \
           -o i18n/translations.po src/*.py
```

Compile po files:

```
$ msgfmt -v translations.po -o translations.mo
```

Custom Rules

It is easy to provide your own validation rule. The rule is any callable with the following contract:

```
def check(self, value, name, model, result, gettext):
```

Here is a description of each attribute:

- `value` - value that is currently validated.

- `name` - name of attribute.
- `model` - object being validated.
- `result` - a dictionary that accepts validation errors.
- `gettext` - a function used to provide i18n support.

2.3.3 Validation Mixin

`ValidationMixin` provides a sort of contextual integration with third party modules. Specifically this mixin requires two attributes: `errors` and `translations`. Once these two attributes provided, validation can be simplified. Let's review it by example:

```
user_validator = Validator({
    'name': [required]
})
```

We defined `user_validator`. Now here is our integration in some service:

```
class MyService(ValidationMixin):

    def __init__(self):
        self.errors = {}
        self.translations = {'validation': None}

    def signin(self, user):
        succeed = self.validate(user, user_validator)
        ...
        self.error('Unauthorized')
        return False
```

If the `signin` operation fails the client can request all validation errors from `errors` attribute. Note that general error message ('Unauthorized') is stored under `__ERROR__` key. Thus it can be used to display general information to the end user.

2.3.4 Model Update

Web form submit is a dictionary where key is the name of the input element being submitted and value is a list. That list can have just a single value for elements like input or several values that depict user choice.

`try_update_model()` method is provided to try update any given object with values submitted by web form.

The convention used by *`try_update_model()`* method is requirement for the model to be properly initialized with default values, e.g. integer attributes must default to some integer value, etc.

List of supported `value_providers`:

```
                try:
                    return date(*strptime(value, fmt)[:3])
                except ValueError:
                    continue
            raise ValueError()
    else:
        return None
```

(continues on next page)

(continued from previous page)

```
def time_value_provider(value, gettext):
    """Converts ``value`` to ``datetime.time``."""
```

Example of domain model initialized with defaults:

```
class Credential(object):

    def __init__(self):
        self.username = ''
        self.password = ''
```

Values submitted by web form:

```
values = {'username': [''], 'password': ['']}
```

Typical use case as follows:

```
from wheezy.validation.model import try_update_model

credential = Credential()
errors = {}
succeed = try_update_model(credential, values, errors)
```

`errors` dictionary contains all errors reported during model update. Key corresponds to attribute being updated, while value is a list of errors.

Numbers

Number value providers (*int_value_provider()*, *decimal_value_provider()*, *float_value_provider()*) supports thousands separator as well as decimal separator. Take a look at the `validation.po` file.

Date and Time

Date and time value providers (*date_value_provider()*, *time_value_provider()*, *datetime_value_provider()*) support a number of formats. Generally there is a default format and fallback formats. It tries the default format and if it fails tries fallback formats. Take a look at the `validation.po` file for a list of supported formats.

Please note that *datetime_value_provider()* falls back to *date_value_provider()* in case none of its own formats matched. Empty value is converted to minimal value for date/time.

Lists

try_update_model() method supports list attributes. Note that existing model list is used (it is not overwritten).

```
>>> class User(object):
...     def __init__(self):
...         self.prefs = []
...         self.prefs2 = [0]
>>> user = User()
>>> values = {'prefs': ['1', '2'], 'prefs2': ['1', '2']}
>>> results = {}
```

(continues on next page)

(continued from previous page)

```
>>> try_update_model(user, values, results)
True
>>> user.prefs
['1', '2']
>>> user.prefs2
[1, 2]
```

Note that the type of the first element in the list selects value_provider for all elements in the list.

Custom Value Providers

Value provider is any callable of the following contract:

```
def my_value_provider(str_value, gettext):
    return parsed_value
```

You can add your value provider to defaults:

```
from wheezy.validation.model import value_providers

value_providers['my_type'] = my_value_provider
```

2.4 Modules

2.4.1 wheezy.validation

class wheezy.validation.**ValidationMixin**

Used primary by service layer to validate domain model.

Requirements: - self.errors - self.translations

Example:

```
class MyService(ValidationMixin):

    def __init__(self, repository, errors, translations, locale):
        # ...

    def authenticate(self, credential):
        if not self.validate(credential, credential_validator):
            return False
        # ...
        return True
```

error(*message*, *name='__ERROR__'*)

Add *message* to errors.

validate(*model*, *validator*)

Validate given *model* using *validator*.

wheezy.validation.**try_update_model**(*model*, *values*, *results*, *translations=None*)

Try update *model* with *values* (a dict of lists or strings), any errors encountered put into *results* and use *translations* for i18n.

class wheezy.validation.**Validator**(*mapping*)
Container of validation rules that all together provide object validation.

validate(*model*, *results*, *stop=True*, *translations=None*, *gettext=None*)
Validates given *model* with results of validation stored in *results*. Be default the validation stops on first rule fail, however with supplied *stop* argument set *False* the *result* will get all errors reported by a rule.

There is a way to internationalize validation errors with *translations* or *gettext*.

2.4.2 wheezy.validation.checker

checker module.

class wheezy.validation.checker.**Checker**(*stop=True*, *translations=None*, *gettext=None*)
Intended to be used by unittest/doctest for validation rules. It is recommended to use test case per validator, test method per attribute, split by success check first than fails.

check(***kwargs*)
Returns a result of validation limited to attributes in *kwargs* which represents attributes of model being validated.

error(***kwargs*)
Returns first error reported by validator.

errors(***kwargs*)
Returns all errors reported by validator.

use(*validator*)
Use *validator* for next series of checks.

class wheezy.validation.checker.**Model**
Simulate plain python class, read-only dictionary access through attributes.

2.4.3 wheezy.validation.i18n

i18n module.

2.4.4 wheezy.validation.mixin

mixin module.

class wheezy.validation.mixin.**ErrorsMixin**
Used primary by service layer to validate business rules.

Requirements: - self.errors

Example:

```
class MyService(ValidationMixin):

    def __init__(self, repository, errors, locale):
        # ...

    def authenticate(self, credential):
        if not self.factory.membership.authenticate(credentials):
            self.error('The username or password provided '
                       'is incorrect.')
```

(continues on next page)

(continued from previous page)

```
            return False
        # ...
        return True
```

error(*message*, *name='__ERROR__'*)
Add *message* to errors.

class wheezy.validation.mixin.**ValidationMixin**
Used primary by service layer to validate domain model.

Requirements: - self.errors - self.translations

Example:

```
class MyService(ValidationMixin):

    def __init__(self, repository, errors, translations, locale):
        # ...

    def authenticate(self, credential):
        if not self.validate(credential, credential_validator):
            return False
        # ...
        return True
```

error(*message*, *name='__ERROR__'*)
Add *message* to errors.

validate(*model*, *validator*)
Validate given *model* using *validator*.

2.4.5 wheezy.validation.model

`model` module.

wheezy.validation.model.**bool_value_provider**(*value*, *gettext*)
Converts `value` to `bool`.

wheezy.validation.model.**bytes_value_provider**(*value*, *gettext*)
Converts `value` to `bytes`.

wheezy.validation.model.**date_value_provider**(*value*, *gettext*)
Converts `value` to `datetime.date`.

wheezy.validation.model.**datetime_value_provider**(*value*, *gettext*)
Converts `value` to `datetime.datetime`.

wheezy.validation.model.**decimal_value_provider**(*value*, *gettext*)
Converts `value` to `Decimal`.

wheezy.validation.model.**float_value_provider**(*value*, *gettext*)
Converts `value` to `float`.

wheezy.validation.model.**int_value_provider**(*value*, *gettext*)
Converts `value` to `int`.

wheezy.validation.model.**str_value_provider**(*value*, *gettext*)
Converts `value` to `str`.

wheezy.validation.model.**time_value_provider**(*value*, *gettext*)
Converts value to datetime.time.

wheezy.validation.model.**try_update_model**(*model*, *values*, *results*, *translations=None*)
Try update *model* with *values* (a dict of lists or strings), any errors encountered put into *results* and use *translations* for i18n.

2.4.6 wheezy.validation.patches

patches module.

wheezy.validation.patches.**patch_strptime_cache_size**(*max_size=100*)
Patch for strptime regex cache max size.

wheezy.validation.patches.**patch_use_cdecimal**()
Use cdecimal module globally. Pure python implementation in-place replacement.

2.4.7 wheezy.validation.rules

rules module.

class wheezy.validation.rules.**AdapterRule**(*converter*, *rule*, *message_template=None*)
Adapts value according to converter. This is useful when you need keep string input in model but validate as an integer.

class wheezy.validation.rules.**AndRule**(**rules*)
Applies all rules regardless of validation result.

validate(*value*, *name*, *model*, *result*, *gettext*)
Iterate over each rule and check whenever any item in value fail.

value - iteratable.

class wheezy.validation.rules.**Base64Rule**(*altchars='+/'*, *message_template=None*)
Ensures a valid base64 string input.

class wheezy.validation.rules.**CompareRule**(*equal=None*, *not_equal=None*, *message_template=None*)
Compares attribute being validated with some other attribute value.

class wheezy.validation.rules.**EmailRule**(*message_template=None*)
Ensures a valid email.

class wheezy.validation.rules.**IgnoreRule**(**args*, ***kwargs*)
The idea behind this rule is to be able to substitute any validation rule by this one that always succeed:

from wheezy.validation.rules import ignore as regex

This way all *regex* rules are ignored within a scope of import.

validate(*value*, *name*, *model*, *result*, *gettext*)
Always succeed.

class wheezy.validation.rules.**IntAdapterRule**(*rule*, *message_template=None*)
Adapts value to an integer.

class wheezy.validation.rules.**IteratorRule**(*rules*, *stop=True*)
Applies rules to each item in value list.

validate(*value*, *name*, *model*, *result*, *gettext*)
Iterate over each rule and check whenever any item in value fail.

`value` - iteratable.

class wheezy.validation.rules.**LengthRule**(*min=None*, *max=None*, *message_template=None*)
Result of python function `len()` must fall within a range defined by this rule.

class wheezy.validation.rules.**MissingRule**(*message_template=None*)
Any value evaluated to boolean `False` pass this rule.

class wheezy.validation.rules.**NotNoneRule**(*message_template=None*)
None value will not pass this rule.

class wheezy.validation.rules.**OneOfRule**(*items*, *message_template=None*)
Value must match at least one element from `items`. Checks are case sensitive if items are strings.

validate(*value*, *name*, *model*, *result*, *gettext*)
Check whenever value belongs to `self.items`.

class wheezy.validation.rules.**OrRule**(**rules*)
Succeed if at least one rule in `rules` succeed.

validate(*value*, *name*, *model*, *result*, *gettext*)
Iterate over each rule and check whenever value fail. Stop on first succeed.

class wheezy.validation.rules.**PredicateRule**(*predicate*, *message_template=None*)
Fails if predicate return False. Predicate is any callable of the following contract:

```
def predicate(model):
    return True
```

class wheezy.validation.rules.**RangeRule**(*min=None*, *max=None*, *message_template=None*)
Ensures value is in range defined by this rule.

Works with any numbers including *Decimal*.

class wheezy.validation.rules.**RegexRule**(*regex*, *negated=False*, *message_template=None*)
Search for regular expression pattern.

class wheezy.validation.rules.**RelativeDateDeltaRule**(*min=None*, *max=None*, *message_template=None*)
Check if value is in relative date range local time.

class wheezy.validation.rules.**RelativeDateTimeDeltaRule**(*min=None*, *max=None*, *message_template=None*)
Check if value is in relative datetime range local time.

class wheezy.validation.rules.**RelativeDeltaRule**(*min=None*, *max=None*, *message_template=None*)
Check if value is in relative date/time range.

```
>>> r = RelativeDeltaRule()
>>> r.now() # doctest: +ELLIPSIS
Traceback (most recent call last):
    ...
NotImplementedError: ...
```

class wheezy.validation.rules.**RelativeTZDateDeltaRule**(*min=None*, *max=None*, *tz=None*, *message_template=None*)
Check if value is in relative date range TZ time.

class wheezy.validation.rules.**RelativeTZDateTimeDeltaRule**(*min=None*, *max=None*, *tz=None*, *message_template=None*)
Check if value is in relative date range TZ time.

class wheezy.validation.rules.**RelativeUTCDateDeltaRule**(*min=None*, *max=None*, *message_template=None*)
Check if value is in relative date range UTC time.

class wheezy.validation.rules.**RelativeUTCDateTimeDeltaRule**(*min=None*, *max=None*, *message_template=None*)
Check if value is in relative datetime range UTC time.

class wheezy.validation.rules.**RelativeUnixTimeDeltaRule**(*min=None*, *max=None*, *message_template=None*)
Check if value is in relative unix range local time.

class wheezy.validation.rules.**RequiredRule**(*message_template=None*)
Any value evaluated to boolean `True` pass this rule. You can extend this validator by supplying additional false values to `required_but_missing` list.

class wheezy.validation.rules.**ScientificRule**(*message_template=None*)
Ensures a valid scientific string input.

class wheezy.validation.rules.**SlugRule**(*message_template=None*)
Ensures only letters, numbers, underscores or hyphens.

class wheezy.validation.rules.**URLSafeBase64Rule**(*message_template=None*)
Ensures a valid base64 URL-safe string input using an alphabet, which substitutes - instead of + and _ instead of / in the standard Base64 alphabet. The input can still contain =.

class wheezy.validation.rules.**ValuePredicateRule**(*predicate*, *message_template=None*)
Fails if predicate return False. Predicate is any callable of the following contract:

```
def predicate(value):
    return True
```

wheezy.validation.rules.**adapter**
alias of *wheezy.validation.rules.AdapterRule*

wheezy.validation.rules.**and_**
alias of *wheezy.validation.rules.AndRule*

wheezy.validation.rules.**compare**
alias of *wheezy.validation.rules.CompareRule*

wheezy.validation.rules.**ignore**
alias of *wheezy.validation.rules.IgnoreRule*

wheezy.validation.rules.**int_adapter**
alias of *wheezy.validation.rules.IntAdapterRule*

wheezy.validation.rules.**iterator**
alias of *wheezy.validation.rules.IteratorRule*

wheezy.validation.rules.**length**
alias of *wheezy.validation.rules.LengthRule*

wheezy.validation.rules.**model_predicate**
alias of *wheezy.validation.rules.PredicateRule*

wheezy.validation.rules.**must**
alias of *wheezy.validation.rules.ValuePredicateRule*

wheezy.validation.rules.**one_of**
alias of *wheezy.validation.rules.OneOfRule*

wheezy.validation.rules.**or_**
alias of *wheezy.validation.rules.OrRule*

wheezy.validation.rules.**predicate**
alias of *wheezy.validation.rules.PredicateRule*

wheezy.validation.rules.**range**
alias of *wheezy.validation.rules.RangeRule*

wheezy.validation.rules.**regex**
alias of *wheezy.validation.rules.RegexRule*

wheezy.validation.rules.**relative_date**
alias of *wheezy.validation.rules.RelativeDateDeltaRule*

wheezy.validation.rules.**relative_datetime**
alias of *wheezy.validation.rules.RelativeDateTimeDeltaRule*

wheezy.validation.rules.**relative_timestamp**
alias of *wheezy.validation.rules.RelativeUnixTimeDeltaRule*

wheezy.validation.rules.**relative_tzdate**
alias of *wheezy.validation.rules.RelativeTZDateDeltaRule*

wheezy.validation.rules.**relative_tzdatetime**
alias of *wheezy.validation.rules.RelativeTZDateTimeDeltaRule*

wheezy.validation.rules.**relative_unixtime**
alias of *wheezy.validation.rules.RelativeUnixTimeDeltaRule*

wheezy.validation.rules.**relative_utcdate**
alias of *wheezy.validation.rules.RelativeUTCDateDeltaRule*

wheezy.validation.rules.**relative_utcdatetime**
alias of *wheezy.validation.rules.RelativeUTCDateTimeDeltaRule*

wheezy.validation.rules.**value_predicate**
alias of *wheezy.validation.rules.ValuePredicateRule*

2.4.8 wheezy.validation.validator

`validator` module.

class wheezy.validation.validator.**Validator**(*mapping*)
Container of validation rules that all together provide object validation.

validate(*model*, *results*, *stop=True*, *translations=None*, *gettext=None*)
Validates given *model* with results of validation stored in *results*. Be default the validation stops on first rule fail, however with supplied *stop* argument set *False* the *result* will get all errors reported by a rule.

There is a way to internationalize validation errors with *translations* or *gettext*.

Python Module Index

w

Index

P

R

S

T

U

V

W