Python

Apr 09, 2020

Contents

[1]:
```
    from wtypes import *
    import pytest, anyconfig
```

[2]:
```
    def test_load_config():
        class BadProject(Bunch):
            tool: Dict
            __annotations__ = {'build-system': Integer}

        with pytest.raises(ValidationError):
            BadProject().load('pyproject.toml')

        class PyProject(Bunch):
            tool: Dict
            __annotations__ = {'build-system': Dict}
        PyProject().load('pyproject.toml')
```

[3]:
```
    def test_dict():
        class Thing(Dict):
            a: Integer

        assert Thing._schema.toDict() == {'type': 'object', 'properties': {'a': {'type
↪': 'integer'}}}
        assert Thing(a=1)
        with pytest.raises(ValidationError):
            Thing(a='abc')
```

[4]:
```
    def test_bunch():
        class Thing(Bunch):
            a: Integer

        assert Thing._schema.toDict() == {'type': 'object', 'properties': {'a': {'type
↪': 'integer'}}}
        t = Thing(a=1)
        assert t.a == 1
        with pytest.raises(ValidationError):
            Thing(a='abc')
```

[5]:
```
    def test_dataclass():
        class Thing(DataClass):
            a: Integer

        assert dataclasses.is_dataclass(Thing)
        assert Thing._schema.toDict() == {'type': 'object', 'properties': {'a': {'type
↪': 'integer'}}}
        assert Thing(a=1)
        with pytest.raises(ValidationError):
            Thing(a='abc')
```

This page contains auto-generated API reference documentation[1].

[1] Created with sphinx-autoapi

CHAPTER 1

wtypes

extended python types for the web and json

Notes

wtypes.**simpleTypes**
The limited set of base types provide by jsonschema.

Type list

1.1 Module Contents

wtypes.**__version__ = 0.0.1**

wtypes.**simpleTypes**

wtypes.**ValidationError**

wtypes.**istype**(*object*, *cls*)
instance(object, type) and issubclass(object, cls)

Examples

```
>>> assert istype(int, int)
>>> assert not istype(10, int)
```

class wtypes.**_NoTitle**
A subclass suppresses the class name when combining schema

class wtypes.**_NoInit**
A subclass to restrict initializing an object from the type.

class wtypes.**_SchemaMeta**
Bases: abc.ABCMeta

Meta operations for wtypes.

The _SchemaMeta ensures that a type's extended schema is validate. Types cannot be generated with invalid schema.

_meta_schema
The schema the type system validates against.

Type dict

_schema
The schema the object validates against.

Type dict

_type
The resource type.

_meta_schema

_schema

_type = http://json-schema.org/draft-07/schema#/properties/

_merge_annotations(*cls*)
Merge annotations from the module resolution order.

_merge_schema(*cls*)
Merge schema from the module resolution order.

create(*cls*, *name: str*, ***schema*)
Create a new schema type.

Parameters

- **name** (*str*) – The title of the new type/schema
- ****schema** (*dict*) – Extra features to add include in the schema.

Returns

Return type type

__neg__(*cls*)
The Not version of a type.

__pos__(*cls*)
The type.

__add__(*cls*, *object*)
Add types together

__and__(*cls*, *object*)
AllOf the conditions

__sub__(*cls*, *object*)
AnyOf the conditions

__or__(*cls*, *object*)
OneOf the conditions

validate(*cls*, *object*)
Validate an object against type's schema.

Note: `isinstance` can used for validation, too.

Parameters **`object`** – An object to validate.

Raises `jsonschema.ValidationError` – The `jsonschema` module validation throws an exception on failure, otherwise the returns a None type.

`__instancecheck__`(*cls*, *object*)

class wtypes.**_ConstType**
Bases: *wtypes._SchemaMeta*

ConstType permits bracketed syntax for defining complex types.

Note: The bracketed notebook should differeniate actions on types versus those on objects.

`__getitem__`(*cls*, *object*)

class wtypes.**_ContainerType**
Bases: *wtypes._ConstType*

ContainerType extras schema from bracketed arguments to define complex types.

`__getitem__`(*cls*, *object*)

wtypes.**_python_to_wtype**(*object*)

wtypes.**_get_schema_from_typeish**(*object*)
infer a schema from an object.

wtypes.**_lower_key**(*str*)

wtypes.**_object_to_webtype**(*object*)

wtypes.**_construct_title**(*cls*)

class wtypes.**Trait**
A trait is an object validated by a validate `jsonschema`.

`_schema`

`_type = http://www.w3.org/2000/01/rdf-schema#Resource`

class wtypes.**Description**
Bases: *wtypes._NoInit*, *wtypes.Trait*, *wtypes._NoTitle*

An empty type with a description

Examples

```
>>> yo = Description['yo']
>>> yo._schema.toDict()
{'description': 'yo'}
```

class wtypes.**Examples**
Bases: *wtypes._NoInit*, *wtypes.Trait*

class wtypes.**Title**

Bases: *wtypes._NoInit*, *wtypes.Trait*, *wtypes._NoTitle*

An empty type with a title

Examples

```
>>> holla = Title['holla']
>>> holla._schema.toDict()
{'title': 'holla'}
```

class wtypes.**Const**

Bases: *wtypes._NoInit*, *wtypes.Trait*

A constant

Examples

```
>>> Const[10]._schema.toDict()
{'const': 10}
```

```
>>> assert isinstance('thing', Const['thing'])
>>> assert not isinstance('jawn', Const['thing']), "Because the compiler is from␣
↪Philly."
```

class wtypes.**Bool**

Bases: *wtypes.Trait*

Boolean type

Examples

```
>>> Bool(), Bool(True), Bool(False)
(False, True, False)
```

Note: It is not possible to base class `bool` so object creation is customized.

_schema

class wtypes.**Null**

Bases: *wtypes.Trait*

nil, none, null type

Examples

```
>>> Null(None)
```

_schema

class wtypes.**_NumericSchema**
Bases: *wtypes._SchemaMeta*

Meta operations for numerical types

__rgt__

__rge__

__rlt__

__rle__

__ge__(*cls*, *object*)
Inclusive minimum

__gt__(*cls*, *object*)
Exclusive minimum

__le__(*cls*, *object*)
Inclusive maximum

__lt__(*cls*, *object*)
Exclusive maximum

__truediv__(*cls*, *object*)
multiple of a number

class wtypes.**Integer**
Bases: *wtypes.Trait*, int

integer type

```
>>> assert isinstance(10, Float)
```

Symbollic conditions.

```
>>> bounded = (10< Float)< 100
>>> bounded._schema.toDict()
{'type': 'number', 'exclusiveMinimum': 10, 'exclusiveMaximum': 100}
```

```
>>> assert isinstance(12, bounded)
>>> assert not isinstance(0, bounded)
```

Multiples

```
>>> assert (Integer+MultipleOf[3])(9) == 9
```

_schema

class wtypes.**Float**
Bases: *wtypes.Trait*, float

float type

```
>>> assert isinstance(10, Integer)
>>> assert not isinstance(10.1, Integer)
```

```
>>> bounded = (10< Float)< 100
>>> bounded._schema.toDict()
{'type': 'number', 'exclusiveMinimum': 10, 'exclusiveMaximum': 100}
```

```
>>> assert isinstance(12, bounded)
>>> assert not isinstance(0, bounded)
>>> assert (Integer/3)(9) == 9
```

_schema

class wtypes.**MultipleOf**
Bases: *wtypes._NoInit*, *wtypes.Trait*

A multipleof constraint for numeric types.

class wtypes.**Minimum**
Bases: *wtypes._NoInit*, *wtypes.Trait*

A minimum constraint for numeric types.

class wtypes.**ExclusiveMinimum**
Bases: *wtypes._NoInit*, *wtypes.Trait*

A exclusive minimum constraint for numeric types.

class wtypes.**Maximum**
Bases: *wtypes._NoInit*, *wtypes.Trait*

A exclusive maximum constraint for numeric types.

class wtypes.**ExclusiveMaximum**
Bases: *wtypes._NoInit*, *wtypes.Trait*

A exclusive maximum constraint for numeric types.

class wtypes.**Properties**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Object properties.

class wtypes.**_ObjectSchema**
Bases: *wtypes._SchemaMeta*

Meta operations for the object schema.

__getitem__(*cls*, *object*)

class wtypes.**_Object**
Base class for validating object types.

_schema

classmethod __init_subclass__(*cls*, ***kwargs*)

load(*self*, **object*)

class wtypes.**Dict**(**args*, ***kwargs*)
Bases: *wtypes.Trait*, dict, *wtypes._Object*

dict type

Examples

```
>>> assert isinstance({}, Dict)
>>> assert not isinstance([], Dict)
```

```
>>> assert isinstance({'a': 1}, Dict + Required['a',])
>>> assert not isinstance({}, Dict + Required['a',])
```

```
>>> assert not isinstance({'a': 'b'}, Dict[Integer, Float])
>>> assert Dict[Integer]({'a': 1}) == {'a': 1}
```

```
>>> Dict[{'a': int}]._schema.toDict()
{'type': 'object', 'properties': {'a': {'type': 'integer'}}}
>>> Dict[{'a': int}]({'a': 1})
{'a': 1}
```

class wtypes.**Bunch**(**args*, ***kwargs*)

Bases: *wtypes.Dict*, munch.Munch

Bunch type

Examples

```
>>> Bunch[{'a': int}]._schema.toDict()
{'type': 'object', 'properties': {'a': {'type': 'integer'}}}
>>> Bunch[{'a': int}]({'a': 1}).toDict()
{'a': 1}
```

class wtypes.**DataClass**

Bases: *wtypes.Trait*, *wtypes._Object*

Validating dataclass type

Examples

```
>>> class q(DataClass): a: int
>>> q._schema.toDict()
{'type': 'object', 'properties': {'a': {'type': 'integer'}}}
```

```
>>> q(a=10)
q(a=10)
```

classmethod **__init_subclass__**(*cls*, ***kwargs*)

__post_init__(*self*)

class wtypes.**AdditionalProperties**

Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Additional object properties.

class wtypes.**Required**

Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Required properties.

class wtypes.**minProperties**

Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Minimum number of properties.

class wtypes.**maxProperties**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Maximum number of properties.

class wtypes.**PropertyNames**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Propery name constraints.

class wtypes.**Dependencies**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Properties dependencies.

class wtypes.**PatternProperties**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Pattern properties names.

class wtypes.**_StringSchema**
Bases: *wtypes._SchemaMeta*

Meta operations for strings types.

__mod__(*cls*, *object*)
A pattern string type.

__gt__(*cls*, *object*)
Minumum string length

__lt__(*cls*, *object*)
Maximum string length

class wtypes.**String**
Bases: *wtypes.Trait*, str

string type.

Examples

```
>>> assert isinstance('abc', String)
```

String patterns

```
>>> assert isinstance('abc', String%"^a")
>>> assert not isinstance('abc', String%"^b")
```

String constraints

```
>>> assert isinstance('abc', (2<String)<10)
>>> assert not isinstance('a', (2<String)<10)
>>> assert not isinstance('a'*100, (2<String)<10)
```

_schema

class wtypes.**MinLength**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Minimum length of a string type.

class wtypes.**MaxLength**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Maximum length of a string type.

class wtypes.**ContentMediaType**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Content type of a string.

class wtypes.**Pattern**
Bases: *wtypes.Trait*, *wtypes._NoInit*

A regular expression pattern.

class wtypes.**_ListSchema**
Bases: *wtypes._SchemaMeta*

Meta operations for list types.

__getitem__(*cls*, *object*)

class wtypes.**List**
Bases: *wtypes.Trait*, list

List type

Examples

List

```
>>> assert isinstance([], List)
>>> assert not isinstance({}, List)
```

Typed list

```
>>> assert isinstance([1], List[Integer])
>>> assert not isinstance([1.1], List[Integer])
```

Tuple

```
>>> assert isinstance([1, '1'], List[Integer, String])
>>> assert not isinstance([1, {}], List[Integer, String])
```

_schema

class wtypes.**Unique**
Bases: *wtypes.List*

Unique list type

Examples

```
>>> assert isinstance([1,2], Unique)
>>> assert not isinstance([1,1], Unique)
```

class wtypes.**_TupleSchema**
Bases: *wtypes._SchemaMeta*

Meta operations for list types.

__getitem__(*cls*, *object*)

class wtypes.**Tuple**
Bases: *wtypes.Trait*

tuple type

Note: There are no tuples in json, they are typed lists.

```
>>> assert Tuple._schema == List._schema
```

Examples

```
>>> assert isinstance([1,2], Tuple)
>>> assert isinstance([1,'1'], Tuple[Integer, String])
>>> assert not isinstance([1,1], Tuple[Integer, String])
```

_schema

class wtypes.**UniqueItems**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

Schema for unique items in a list.

class wtypes.**Contains**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

class wtypes.**Items**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

class wtypes.**AdditionalItems**
Bases: *wtypes.Trait*, *wtypes._NoInit*, *wtypes._NoTitle*

class wtypes.**Not**
Bases: *wtypes.Trait*

not schema.

Examples

```
>>> assert Not[String](100) == 100
>>> assert not isinstance('abc', Not[String])
```

Note: See the __neg__ method for symbollic not composition.

---

class wtypes.**AnyOf**
Bases: *wtypes.Trait*, *wtypes._NoInit*

anyOf combined schema.

Examples

```
>>> assert isinstance(10, AnyOf[Integer, String])
>>> assert not isinstance([], AnyOf[Integer, String])
```

class wtypes.**AllOf**
Bases: *wtypes.Trait*, *wtypes._NoInit*

allOf combined schema.

Examples

```
>>> assert isinstance(9, AllOf[Float>0, Integer/3])
>>> assert not isinstance(-9, AllOf[Float>0, Integer/3])
```

class wtypes.**OneOf**
Bases: *wtypes.Trait*, *wtypes._NoInit*

oneOf combined schema.

Examples

```
>>> assert isinstance(-9, OneOf[Float>0, Integer/3])
>>> assert not isinstance(9, OneOf[Float>0, Integer/3])
```

class wtypes.**Enum**
Bases: *wtypes.Trait*, *wtypes._NoInit*

An enumerate type that is restricted to its inputs.

Examples

```
>>> assert isinstance('cat', Enum['cat', 'dog'])
>>> assert not isinstance('', Enum['cat', 'dog'])
```

wtypes.**_formats**

class wtypes.**ContentEncoding**
Bases: Enum['7bit 8bit binary quoted-printable base64'.split()], *wtypes._NoInit*, *wtypes._NoTitle*

Content encodings for a string.

class wtypes.**Format**
Bases: Enum[_formats], *wtypes._NoInit*, *wtypes._NoTitle*

wtypes.**compile**

class wtypes.**If**
Bases: *wtypes.Trait*

class wtypes.**Then**
Bases: *wtypes.Trait*

class wtypes.**Else**
Bases: *wtypes.Trait*

class wtypes.**Configurable**
Bases: *wtypes.DataClass*

A configurable classs that is create with dataclass syntax.

CHAPTER 2

conf

2.1 Module Contents

conf.**master_doc = index**

conf.**extensions**

conf.**autoapi_type = python**

conf.**autoapi_dirs = ['.']**

conf.**SUFFIX**

conf.**html_static_path = []**

conf.**using_rtd_theme**

conf.**theme**

conf.**websupport2_base_url = https://readthedocs.org/websupport**

conf.**context**

conf.**html_context**

conf.**extensions = ['readthedocs_ext.readthedocs']**

conf.**project_language = en**

conf.**language_user**

conf.**latex_engine_user**

conf.**latex_elements_user**

conf.**latex_use_xindy = False**

conf.**chinese**

conf.**japanese**

conf.**latex_engine**

CHAPTER 3

setup

3.1 Module Contents

setup.**flit**

setup.**metadata**

Python Module Index

Index

Symbols

A

B

C

D

E

F

H

I

J

L

M

N

O

P

R

S

T

U

V

W