Python

Apr 17, 2020

Contents

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

[1] Created with sphinx-autoapi

CHAPTER 1

conf

1.1 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 2

setup

2.1 Module Contents

setup.**flit_data**

setup.**metadata**

CHAPTER 3

conftest

CHAPTER 4

wtypes

extended python types for the web and json

Notes

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

Type list

4.1 Subpackages

4.1.1 wtypes.widgets

Widgets features for wtypes.

Submodules

wtypes.widgets.ipywidgets

4.2 Submodules

4.2.1 wtypes.base

An extended type and trait system for python.

Notes

Module Contents

wtypes.base.**__version__ = 0.0.1**

wtypes.base.**ValidationError**

class wtypes.base.**_Implementation**
An implementation of the pluggy wtypes spec.

Notes

The implementation needs to be registered with the plugin manager.

validate_type(*type*)

validate_object(*object*, *schema*)

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

Examples

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

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

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

class wtypes.base.**_ContextMeta**
Bases: abc.ABCMeta

Meta operations for the context..

_context
The schema the object validates against.

Type dict

_type
The schema the object validates against.

Type type

_schema

Type dict

Notes

A context type cannot be verified as it only describes, although some descriptors like SHACL can validate

_schema

_context

_type

_merge_args(*cls*)

_merge_context(*cls*)

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

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

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

__matmul__(*cls*, *object*)

validate(*cls*, *object*)
A context type does not validate.

__instancecheck__(*cls*, *object*)

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

__add__(*cls*, *object*)

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

__pos__(*cls*)
The type.

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

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

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

class wtypes.base.**_SchemaMeta**
Bases: *wtypes.base._ContextMeta*

Meta operations for wtypes.

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

_schema
The schema the object validates against.

Type dict

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

__pos__(*cls*)
The type.

__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.

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

ContainerType extras schema from bracketed arguments to define complex types.

__getitem__(*cls*, *object*)

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

wtypes.base.**_get_schema_from_typeish**(*object*, *key='anyOf'*)
infer a schema from an object.

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

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

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

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

_schema

_context

classmethod _resolve_defaults(*cls*, **args*, ***kwargs*)

wtypes.base.**get_jawn**(*thing*, *key*, *object*)

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

An empty type with a description

Examples

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

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

class wtypes.base.**Default**
Bases: *wtypes.base._NoInit*, *wtypes.base.Trait*

class wtypes.base.**Title**
Bases: *wtypes.base._NoInit*, *wtypes.base.Trait*, *wtypes.base._NoTitle*

An empty type with a title

Examples

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

class wtypes.base.**Const**
Bases: *wtypes.base._NoInit*, *wtypes.base.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.base.**Bool**
Bases: *wtypes.base.Trait*

Boolean type

Examples

```
>>> Bool(), Bool(True), Bool(False)
(False, True, False)
>>> assert (Bool + Default[True])()
```

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

class wtypes.base.**Null**
Bases: *wtypes.base.Trait*

nil, none, null type

Examples

```
>>> Null(None)
>>> assert (Null + Default[None])() is None
```

class wtypes.base.**_NumericSchema**
Bases: *wtypes.base._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.base.**Integer**
Bases: *wtypes.base.Trait*, `int`

integer type

Examples

```
>>> assert isinstance(10, Integer)
>>> assert not isinstance(10.1, Integer)
>>> (Integer+Default[9])(9)
9
```

```
>>> bounded = (10< Integer)< 100
>>> bounded._schema.toDict()
{'type': 'integer', 'exclusiveMinimum': 10, 'exclusiveMaximum': 100}
>>> assert isinstance(12, bounded)
```

(continues on next page)

(continued from previous page)

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

class wtypes.base.**Float**

Bases: *wtypes.base.Trait*, float

float type

```
>>> assert isinstance(10, Float)
>>> assert isinstance(10.1, Float)
```

Symbollic conditions.

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

```
>>> assert isinstance(12.1, bounded)
>>> assert not isinstance(0.1, bounded)
```

Multiples

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

class wtypes.base.**MultipleOf**

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

A multipleof constraint for numeric types.

class wtypes.base.**Minimum**

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

A minimum constraint for numeric types.

class wtypes.base.**ExclusiveMinimum**

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

A exclusive minimum constraint for numeric types.

class wtypes.base.**Maximum**

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

A exclusive maximum constraint for numeric types.

class wtypes.base.**ExclusiveMaximum**

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

A exclusive maximum constraint for numeric types.

class wtypes.base.**Properties**

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

Object properties.

class wtypes.base.**_ObjectSchema**

Bases: *wtypes.base._SchemaMeta*

Meta operations for the object schema.

__getitem__(*cls*, *object*)

Examples

```
>>> Dict[wtypes.Forward[range], int].__annotations__
{'': typing.Union[abc.Forward, int]}
>>> Dict[wtypes.Forward[range], int]._schema.toDict()
{'type': 'object', 'properties': {}, 'additionalProperties': {'anyOf': [{'type
↪': 'integer'}]}}
```

class wtypes.base.**_Object**

Base class for validating object types.

classmethod __init_subclass__(*cls*, ***schema*)

classmethod from_config_file(*cls*, **object*)

class wtypes.base.**Dict**

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

dict type

Examples

```
>>> assert istype(Dict, __import__('collections').abc.MutableMapping)
>>> assert (Dict + Default[{'b': 'foo'}])() == {'b': 'foo'}
>>> assert (Dict + Default[{'b': 'foo'}])({'a': 'bar'}) == {'b': 'foo', 'a': 'bar
↪'}
```

```
>>> 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}
```

__setitem__(*self*, *key*, *object*)

Only test the key being set to avoid invalid state.

update(*self*, **args*, ***kwargs*)

class wtypes.base.**Bunch**

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

Additional object properties.

class wtypes.base.**Required**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

Required properties.

class wtypes.base.**minProperties**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

Minimum number of properties.

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

Maximum number of properties.

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

Propery name constraints.

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

Properties dependencies.

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

Pattern properties names.

class wtypes.base.**_StringSchema**
Bases: *wtypes.base._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.base.**String**
Bases: *wtypes.base.Trait*, str

string type.

Examples

```
>>> assert isinstance('abc', String)
>>> assert (String+Default['abc'])() == 'abc'
```

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)
```

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

Minimum length of a string type.

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

Maximum length of a string type.

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

Content type of a string.

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

A regular expression pattern.

class wtypes.base.**Format**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

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

Meta operations for list types.

__getitem__(*cls*, *object*)
List meta operations for bracketed type notation.

Examples

```
>>> List[wtypes.Forward[range], int]._type
typing.List[typing.Union[abc.Forward, int]]
```

```
>>> Tuple[wtypes.Forward[range], int]._type
typing.Tuple[abc.Forward, int]
```

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

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

class wtypes.base.**List**

Bases: *wtypes.base.Trait*, list

List type

Examples

List

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

Typed list

```
>>> assert List[Integer]([1, 2, 3])
>>> assert isinstance([1], List[Integer])
>>> assert not isinstance([1.1], List[Integer])
```

```
>>> List[Integer, String]._schema.toDict()
{'type': 'array', 'items': {'anyOf': [{'type': 'integer'}, {'type': 'string'}]}}
```

Tuple

```
>>> assert List[Integer, String]([1, 'abc', 2])
>>> assert isinstance([1, '1'], List[Integer, String])
>>> assert not isinstance([1, {}], List[Integer, String])
```

_verify_item(*self*, *object*, *id=None*)

Elemental verification for interactive type checking.

__setitem__(*self*, *id*, *object*)

append(*self*, *object*)

insert(*self*, *id*, *object*)

extend(*self*, *object*)

pop(*self*, *index=-1*)

class wtypes.base.**Unique**

Bases: *wtypes.base.List*

Unique list type

Examples

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

class wtypes.base.**Tuple**

Bases: *wtypes.base.List*

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 Tuple[Integer, String]([1, 'abc'])
>>> Tuple[Integer, String]._schema.toDict()
{'type': 'array', 'items': [{'type': 'integer'}, {'type': 'string'}]}
```

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

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

Schema for unique items in a list.

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

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

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

class wtypes.base.**MinItems**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

Minimum length of an array.

class wtypes.base.**MaxItems**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

Maximum length of an array.

class wtypes.base.**Enum**
Bases: *wtypes.base.Trait*

An enumerate type that is restricted to its inputs.

Examples

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

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

Content encodings for a string.

class wtypes.base.**If**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

if condition type

class wtypes.base.**Then**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

then condition type

class wtypes.base.**Else**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

else condition type

4.2.2 wtypes.combining_types

Module Contents

class wtypes.combining_types.**_NotType**
Bases: *wtypes.base._ConstType*

validate(*cls*, *object*)

__getitem__(*cls*, *object*)

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

not schema.

Examples

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

Note: See the __neg__ method for symbollic not composition.

class wtypes.combining_types.**_AnyOfType**
Bases: *wtypes.base._ConstType*

validate(*cls*, *object*)

__getitem__(*cls*, *object*)

class wtypes.combining_types.**AnyOf**
Bases: *wtypes.base.Trait*

anyOf combined schema.

Examples

```
>>> assert AnyOf[wtypes.Integer, wtypes.String]('abc')
>>> assert isinstance(10, AnyOf[wtypes.Integer, wtypes.String])
>>> assert not isinstance([], AnyOf[wtypes.Integer, wtypes.String])
```

class wtypes.combining_types.**_AllOfType**
Bases: *wtypes.base._ConstType*

validate(*cls*, *object*)

__getitem__(*cls*, *object*)

class wtypes.combining_types.**AllOf**
Bases: *wtypes.base.Trait*

allOf combined schema.

Examples

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

class wtypes.combining_types.**_OneOfType**
Bases: *wtypes.base._ConstType*

validate(*cls*, *object*)

__getitem__(*cls*, *object*)

class wtypes.combining_types.**OneOf**
Bases: *wtypes.base.Trait*

oneOf combined schema.

Examples

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

4.2.3 wtypes.content_types

Module Contents

class wtypes.content_types.**_Content**
Bases: *wtypes.String*

_repr_data_(*self*)

_repr_metadata_(*self*)

_repr_mimebundle_(*self*, *include=None*, *exclude=None*)

class wtypes.content_types.**TextPlain**
Bases: *wtypes.content_types._Content*

class wtypes.content_types.**TextMarkdown**
Bases: *wtypes.content_types.TextPlain*

class wtypes.content_types.**TextHtml**
Bases: *wtypes.content_types.TextPlain*

4.2.4 wtypes.dataclass

Compatability for wtyped dataclasses.

Module Contents

class wtypes.dataclass.**Setter**

__setattr__(*self*, *key*, *object*)
Only test the attribute being set to avoid invalid state.

class wtypes.dataclass.**DataClass**
Bases: *wtypes.dataclass.Setter*, *wtypes.Trait*, *wtypes.base._Object*

Validating dataclass type

Examples

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

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

```
>>> assert not isinstance({}, q)
```

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

4.2.5 wtypes.evented

Evented wtypes, the observable pattern

Module Contents

class wtypes.evented.**spec**

dlink(*this*, *source*, *that*, *target*, *callable*)

link(*this*, *source*, *that*, *target*)

wtypes.evented.**get_jawn**(*thing*, *key*, *object*)

wtypes.evented.**set_jawn**(*thing*, *key*, *object*)

class wtypes.evented.**spec_impl**

__enter__(*self*)

__exit__(*self*, **e*)

class wtypes.evented.**wtypes_impl**
Bases: *wtypes.evented.spec_impl*

dlink(*this*, *source*, *that*, *target*, *callable*)

class wtypes.evented.**Link**

_registered_parents

_registered_links

_registered_id

_deferred_changed

_deferred_prior

_depth = 0

_display_id

__enter__(*self*)

__exit__(*self*, **e*)

link(*this*, *source*, *that*, *target='value'*)

dlink(*self*, *source*, *that*, *target*, *callable=None*)

Examples

```
>>> class d(Dict): a: int
>>> e, f = d(a=1), d(a=1)
>>> e.dlink('a', f, 'a', lambda x: 2*x)
{'a': 1}
>>> e['a'] = 7
>>> f
{'a': 14}
```

observe(*self*, *source=''*, *callable=None*)
The callable has to define a signature.

_propagate(*self*, **changed*, ***prior*)

_update_display(*self*)

_ipython_display_(*self*)

_repr_mimebundle_(*self*, *include=None*, *exclude=None*)

class wtypes.evented.**_EventedObject**
Bases: *wtypes.evented.Link*

class wtypes.evented.**_EventedDict**(**args*, ***kwargs*)
Bases: *wtypes.evented._EventedObject*

_link_parent(*self*, *object*)

__setitem__(*self*, *key*, *object*)

update(*self*, **args*, ***kwargs*)

class wtypes.evented.**_EventedDataClass**(**args*, ***kwargs*)
Bases: *wtypes.evented._EventedObject*

_link_parent(*self*, *object*)

__setattr__(*self*, *key*, *object*)

class wtypes.evented.**_EventedList**(**args*, ***kwargs*)
Bases: *wtypes.evented.Link*

_link_parent(*self*, *object*)

__exit__(*self*, **e*)

__setitem__(*self*, *key*, *object*)

append(*self*, *object*)

insert(*self*, *index*, *object*)

extend(*self*, *object*)

pop(*self*, *index=-1*)

observe(*self*, *callable=None*)
The callable has to define a signature.

class wtypes.evented.**List**(**args*, ***kwargs*)
Bases: *wtypes.evented._EventedList*, wtypes.wtypes.List

class wtypes.evented.**Bunch**(**args*, ***kwargs*)
Bases: *wtypes.evented._EventedDict*, wtypes.wtypes.Bunch

An evented dictionary/bunch

Examples

```
>>> e, f = Bunch(), Bunch()
>>> e.link('a', f, 'b')
Bunch({})
>>> e['a'] = 1
>>> f.toDict()
{'b': 1}
>>> e.update(a=100)
>>> f.toDict()
{'b': 100}
```

```
>>> f['b'] = 2
>>> assert e['a'] == f['b']
>>> e = Bunch().observe('a', print)
>>> e['a'] = 2
{'new': 2, 'old': None, 'object': Bunch({'a': 2}), 'name': 'a'}
```

class wtypes.evented.**Dict**(**args*, ***kwargs*)
Bases: *wtypes.evented._EventedDict*, wtypes.wtypes.Dict

An evented dictionary/bunch

Examples

```
>>> e, f = Dict(), Dict()
>>> e.link('a', f, 'b')
{}
>>> e['a'] = 1
>>> f
{'b': 1}
>>> e.update(a=100)
>>> f
{'b': 100}
```

```
>>> f['b'] = 2
>>> assert e['a'] == f['b']
>>> e = Dict().observe('a', print)
>>> e['a'] = 2
{'new': 2, 'old': None, 'object': {'a': 2}, 'name': 'a'}
```

class wtypes.evented.**DataClass**(**args*, ***kwargs*)
Bases: *wtypes.evented._EventedDataClass*, *wtypes.DataClass*

class wtypes.evented.**Namespace**(**args*, ***kwargs*)
Bases: *wtypes.evented.Dict*

An event namespace to visualize track the annotated fields.

Examples

```
>>> # evented.Namespace.register()
```

_repr_mimebundle_(*self*, *include=None*, *exclude=None*)

classmethod register(*cls*)

classmethod unregister(*cls*)

4.2.6 wtypes.examples

Module Contents

wtypes.examples.**generate_strategy_from_object**(**object*)

wtypes.examples.**generate_strategy_from_type**(*type*)
Generate a hypothesis strategy from a type.

> **Note:** Requires hypothesis_jsonschema

wtypes.examples.**generate_example**(*type*)

wtypes.examples.**example**

4.2.7 wtypes.python_types

Module Contents

class wtypes.python_types.**_NoType**

class wtypes.python_types.**_ForwardSchema**
Bases: *wtypes.base._ContextMeta*

A forward reference to an object, the object must exist in sys.modules.

Notes

Python types live on the __annotations__ attribute.

_type_args

_type_kwargs

__getitem__(*cls*, *object*)

validate(*cls*, *object*)

eval(*cls*)

__add__(*cls*, *object*)

class wtypes.python_types.**_ArgumentSchema**
Bases: *wtypes.python_types._ForwardSchema*

__getitem__(*cls*, *object*)

class wtypes.python_types.**Args**
Bases: *wtypes.python_types._NoType*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

class wtypes.python_types.**Kwargs**
Bases: *wtypes.python_types._NoType*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

class wtypes.python_types.**Forward**
Create type using objects or forward references.

Examples

```
>>> assert Forward['builtins.range']() is range
```

class wtypes.python_types.**Class**
Bases: *wtypes.python_types.Forward*

Create type using objects or forward references.

Examples

```
>>> assert isinstance(range, Class['builtins.range'])
```

classmethod validate(*cls*, *object*)

class wtypes.python_types.**Instance**
Bases: *wtypes.python_types.Forward*

Create an instance of a type using objects or forward references.

Examples

```
>>> assert (Instance[range] + Args[10, 20])() == range(10, 20)
>>> assert (Instance['builtins.range'] + Args[10, 20])() == range(10, 20)
>>> assert isinstance(range(10), Instance['builtins.range'])
```

Deffered references.

```
>>> assert not isinstance(1, Instance['pandas.DataFrame'])
>>> assert 'pandas' not in __import__('sys').modules
```

classmethod validate(*cls*, *object*)

4.2.8 wtypes.rdf

RDF properties from wtypes

Note: Requires rdflib

These are actions based on the _context of the types.

4.2.9 wtypes.spec

Module Contents

wtypes.spec.**specification**

wtypes.spec.**implementation**

wtypes.spec.**manager**

class wtypes.spec.**spec**

validate_type(*type*)
A hook to validate types.

validate_object(*object*, *schema*)
A hook to validate types.

4.2.10 wtypes.string_formats

Module Contents

class wtypes.string_formats.**Color**
Bases: *wtypes.String*

class wtypes.string_formats.**Datetime**
Bases: *wtypes.String*

class wtypes.string_formats.**Time**
Bases: *wtypes.String*

class wtypes.string_formats.**Date**
Bases: *wtypes.String*

class wtypes.string_formats.**Email**
Bases: *wtypes.String*

class wtypes.string_formats.**Idnemail**
Bases: *wtypes.String*

class wtypes.string_formats.**Hostname**
Bases: *wtypes.String*

class wtypes.string_formats.**Idnhostname**
Bases: *wtypes.String*

class wtypes.string_formats.**Ipv4**
Bases: *wtypes.String*

class wtypes.string_formats.**Ipv6**
Bases: *wtypes.String*

class wtypes.string_formats.**Uri**
Bases: *wtypes.String*

get

post

_httpx_method(*self*, *method*, **args*, ***kwargs*)

class wtypes.string_formats.**Urireference**
Bases: *wtypes.String*

class wtypes.string_formats.**Iri**
Bases: *wtypes.string_formats.Uri*

class wtypes.string_formats.**Irireference**
Bases: *wtypes.String*

class wtypes.string_formats.**Uritemplate**
Bases: *wtypes.String*

expand(*self*, ***kwargs*)

URITemplate(*self*)

class wtypes.string_formats.**Jsonpointer**
Bases: *wtypes.String*

resolve_pointer(*self*, *doc*, *default=None*)

class wtypes.string_formats.**Relativejsonpointer**
Bases: *wtypes.String*

class wtypes.string_formats.**Regex**
Bases: *wtypes.String*

4.2.11 wtypes.utils

Module Contents

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

Examples

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

wtypes.utils.**validate_schema**(*object: object*, *schema: dict*) → None

wtypes.utils.**validate_generic**(*object*, *cls*)

4.3 Package Contents

wtypes.**__version__ = 0.0.2**

wtypes.**specification**

wtypes.**implementation**

wtypes.**manager**

class wtypes.**spec**

validate_type(*type*)
A hook to validate types.

validate_object(*object*, *schema*)
A hook to validate types.

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

Examples

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

wtypes.**validate_schema**(*object: object*, *schema: dict*) → None

wtypes.**validate_generic**(*object*, *cls*)

wtypes.**__version__ = 0.0.1**

wtypes.**ValidationError**

class wtypes.**_Implementation**
An implementation of the pluggy wtypes spec.

Notes

The implementation needs to be registered with the plugin manager.

validate_type(*type*)

validate_object(*object*, *schema*)

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.**_ContextMeta**
Bases: abc.ABCMeta

Meta operations for the context..

_context
The schema the object validates against.

Type dict

_type
The schema the object validates against.

Type type

_schema

Type dict

Notes

A context type cannot be verified as it only describes, althrough some descriptors like SHACL can validate

_schema

_context

_type

_merge_args(*cls*)

_merge_context(*cls*)

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

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

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

__matmul__(*cls*, *object*)

validate(*cls*, *object*)
A context type does not validate.

__instancecheck__(*cls*, *object*)

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

__add__ (*cls*, *object*)

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

__pos__ (*cls*)
The type.

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

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

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

class wtypes.**_SchemaMeta**
Bases: *wtypes.base._ContextMeta*

Meta operations for wtypes.

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

_schema
The schema the object validates against.

Type dict

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

__pos__ (*cls*)
The type.

__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.

class wtypes.**_ConstType**
Bases: *wtypes.base._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.base._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*, *key='anyOf'*)
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

_context

classmethod _resolve_defaults(*cls*, **args*, ***kwargs*)

wtypes.**get_jawn**(*thing*, *key*, *object*)

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

An empty type with a description

Examples

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

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

class wtypes.**Default**
Bases: *wtypes.base._NoInit*, *wtypes.base.Trait*

class wtypes.**Title**
Bases: *wtypes.base._NoInit*, *wtypes.base.Trait*, *wtypes.base._NoTitle*

An empty type with a title

Examples

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

class wtypes.**Const**
Bases: *wtypes.base._NoInit*, *wtypes.base.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.base.Trait*

Boolean type

Examples

```
>>> Bool(), Bool(True), Bool(False)
(False, True, False)
>>> assert (Bool + Default[True])()
```

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

class wtypes.**Null**
Bases: *wtypes.base.Trait*

nil, none, null type

Examples

```
>>> Null(None)
>>> assert (Null + Default[None])() is None
```

class wtypes.**_NumericSchema**
Bases: *wtypes.base._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.base.Trait*, int

integer type

Examples

```
>>> assert isinstance(10, Integer)
>>> assert not isinstance(10.1, Integer)
>>> (Integer+Default[9])(9)
9
```

```
>>> bounded = (10< Integer)< 100
>>> bounded._schema.toDict()
{'type': 'integer', 'exclusiveMinimum': 10, 'exclusiveMaximum': 100}
>>> assert isinstance(12, bounded)
>>> assert not isinstance(0, bounded)
>>> assert (Integer/3)(9) == 9
```

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

float type

```
>>> assert isinstance(10, Float)
>>> assert isinstance(10.1, Float)
```

Symbollic conditions.

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

```
>>> assert isinstance(12.1, bounded)
>>> assert not isinstance(0.1, bounded)
```

Multiples

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

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

A multipleof constraint for numeric types.

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

A minimum constraint for numeric types.

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

A exclusive minimum constraint for numeric types.

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

A exclusive maximum constraint for numeric types.

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

A exclusive maximum constraint for numeric types.

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

Object properties.

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

Meta operations for the object schema.

__getitem__(*cls*, *object*)

Examples

```
>>> Dict[wtypes.Forward[range], int].__annotations__
{'': typing.Union[abc.Forward, int]}
>>> Dict[wtypes.Forward[range], int]._schema.toDict()
{'type': 'object', 'properties': {}, 'additionalProperties': {'anyOf': [{'type
↪': 'integer'}]}}
```

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

classmethod __init_subclass__(*cls*, ***schema*)

classmethod from_config_file(*cls*, **object*)

class wtypes.**Dict**
Bases: *wtypes.base.Trait*, dict, *wtypes.base._Object*

dict type

Examples

```
>>> assert istype(Dict, __import__('collections').abc.MutableMapping)
>>> assert (Dict + Default[{'b': 'foo'}])() == {'b': 'foo'}
>>> assert (Dict + Default[{'b': 'foo'}])({'a': 'bar'}) == {'b': 'foo', 'a': 'bar
↪'}
```

```
>>> 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}
```

__setitem__(*self*, *key*, *object*)
Only test the key being set to avoid invalid state.

update(*self*, **args*, ***kwargs*)

class wtypes.**Bunch**
Bases: *wtypes.base.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.**AdditionalProperties**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

Additional object properties.

class wtypes.**Required**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

Required properties.

class wtypes.**minProperties**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

Minimum number of properties.

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

Maximum number of properties.

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

Propery name constraints.

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

Properties dependencies.

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

Pattern properties names.

class wtypes.**_StringSchema**
Bases: *wtypes.base._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.base.Trait*, str

string type.

Examples

```
>>> assert isinstance('abc', String)
>>> assert (String+Default['abc'])() == 'abc'
```

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)
```

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

Minimum length of a string type.

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

Maximum length of a string type.

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

Content type of a string.

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

A regular expression pattern.

class wtypes.**Format**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

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

Meta operations for list types.

__getitem__(*cls*, *object*)
List meta operations for bracketed type notation.

Examples

```
>>> List[wtypes.Forward[range], int]._type
typing.List[typing.Union[abc.Forward, int]]
```

```
>>> Tuple[wtypes.Forward[range], int]._type
typing.Tuple[abc.Forward, int]
```

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

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

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

List type

Examples

List

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

Typed list

```
>>> assert List[Integer]([1, 2, 3])
>>> assert isinstance([1], List[Integer])
>>> assert not isinstance([1.1], List[Integer])
```

```
>>> List[Integer, String]._schema.toDict()
{'type': 'array', 'items': {'anyOf': [{'type': 'integer'}, {'type': 'string'}]}}
```

Tuple

```
>>> assert List[Integer, String]([1, 'abc', 2])
>>> assert isinstance([1, '1'], List[Integer, String])
>>> assert not isinstance([1, {}], List[Integer, String])
```

_verify_item(*self*, *object*, *id=None*)
Elemental verification for interactive type checking.

__setitem__(*self*, *id*, *object*)

append(*self*, *object*)

insert(*self*, *id*, *object*)

extend(*self*, *object*)

pop(*self*, *index=-1*)

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

Unique list type

Examples

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

class wtypes.**Tuple**
Bases: *wtypes.base.List*

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 Tuple[Integer, String]([1, 'abc'])
>>> Tuple[Integer, String]._schema.toDict()
{'type': 'array', 'items': [{'type': 'integer'}, {'type': 'string'}]}
```

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

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

Schema for unique items in a list.

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

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

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

class wtypes.**MinItems**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

Minimum length of an array.

class wtypes.**MaxItems**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

Maximum length of an array.

class wtypes.**Enum**
Bases: *wtypes.base.Trait*

An enumerate type that is restricted to its inputs.

Examples

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

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

Content encodings for a string.

class wtypes.**If**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

if condition type

class wtypes.**Then**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

then condition type

class wtypes.**Else**
Bases: *wtypes.base.Trait*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

else condition type

class wtypes.**_NotType**
Bases: *wtypes.base._ConstType*

validate(*cls*, *object*)

__getitem__(*cls*, *object*)

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

not schema.

Examples

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

Note: See the __neg__ method for symbollic not composition.

class wtypes.**_AnyOfType**
Bases: *wtypes.base._ConstType*

validate(*cls*, *object*)

__getitem__(*cls*, *object*)

class wtypes.**AnyOf**
Bases: *wtypes.base.Trait*

anyOf combined schema.

Examples

```
>>> assert AnyOf[wtypes.Integer, wtypes.String]('abc')
>>> assert isinstance(10, AnyOf[wtypes.Integer, wtypes.String])
>>> assert not isinstance([], AnyOf[wtypes.Integer, wtypes.String])
```

class wtypes.**_AllOfType**
Bases: *wtypes.base._ConstType*

validate(*cls*, *object*)

__getitem__(*cls*, *object*)

class wtypes.**AllOf**
Bases: *wtypes.base.Trait*

allOf combined schema.

Examples

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

class wtypes.**_OneOfType**
Bases: *wtypes.base._ConstType*

validate(*cls*, *object*)

__getitem__(*cls*, *object*)

class wtypes.**OneOf**
Bases: *wtypes.base.Trait*

oneOf combined schema.

Examples

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

class wtypes.**Setter**

__setattr__(*self*, *key*, *object*)
Only test the attribute being set to avoid invalid state.

class wtypes.**DataClass**
Bases: *wtypes.dataclass.Setter*, *wtypes.Trait*, *wtypes.base._Object*

Validating dataclass type

Examples

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

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

```
>>> assert not isinstance({}, q)
```

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

class wtypes.**_NoType**

class wtypes.**_ForwardSchema**
Bases: *wtypes.base._ContextMeta*

A forward reference to an object, the object must exist in sys.modules.

Notes

Python types live on the __annotations__ attribute.

_type_args

_type_kwargs

__getitem__(*cls*, *object*)

validate(*cls*, *object*)

eval(*cls*)

__add__(*cls*, *object*)

class wtypes.**_ArgumentSchema**
Bases: *wtypes.python_types._ForwardSchema*

__getitem__(*cls*, *object*)

class wtypes.**Args**
Bases: *wtypes.python_types._NoType*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

class wtypes.**Kwargs**
Bases: *wtypes.python_types._NoType*, *wtypes.base._NoInit*, *wtypes.base._NoTitle*

class wtypes.**Forward**
Create type using objects or forward references.

Examples

```
>>> assert Forward['builtins.range']() is range
```

class wtypes.**Class**
Bases: *wtypes.python_types.Forward*

Create type using objects or forward references.

Examples

```
>>> assert isinstance(range, Class['builtins.range'])
```

classmethod validate(*cls*, *object*)

class wtypes.**Instance**
Bases: *wtypes.python_types.Forward*

Create an instance of a type using objects or forward references.

Examples

```
>>> assert (Instance[range] + Args[10, 20])() == range(10, 20)
>>> assert (Instance['builtins.range'] + Args[10, 20])() == range(10, 20)
>>> assert isinstance(range(10), Instance['builtins.range'])
```

Deffered references.

```
>>> assert not isinstance(1, Instance['pandas.DataFrame'])
>>> assert 'pandas' not in __import__('sys').modules
```

classmethod validate(*cls*, *object*)

class wtypes.**Color**
Bases: *wtypes.String*

class wtypes.**Datetime**
Bases: *wtypes.String*

class wtypes.**Time**
Bases: *wtypes.String*

class wtypes.**Date**
Bases: *wtypes.String*

class wtypes.**Email**
Bases: *wtypes.String*

class wtypes.**Idnemail**
Bases: *wtypes.String*

class wtypes.**Hostname**
Bases: *wtypes.String*

class wtypes.**Idnhostname**
Bases: *wtypes.String*

class wtypes.**Ipv4**
Bases: *wtypes.String*

class wtypes.**Ipv6**
Bases: *wtypes.String*

class wtypes.**Uri**
Bases: *wtypes.String*

get

post

_httpx_method(*self*, *method*, **args*, ***kwargs*)

class wtypes.**Urireference**
Bases: *wtypes.String*

class wtypes.**Iri**
Bases: *wtypes.string_formats.Uri*

class wtypes.**Irireference**
Bases: *wtypes.String*

class wtypes.**Uritemplate**
Bases: *wtypes.String*

expand(*self*, ***kwargs*)

URITemplate(*self*)

class wtypes.**Jsonpointer**
Bases: *wtypes.String*

resolve_pointer(*self*, *doc*, *default=None*)

class wtypes.**Relativejsonpointer**
Bases: *wtypes.String*

class wtypes.**Regex**
Bases: *wtypes.String*

Python Module Index

C

S

W

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W