
typepy Documentation

Release 0.6.1

Tsuyoshi Hombashi

May 11, 2019

Table of Contents

1	typepy	1
2	Summary	3
3	Features	5
4	Usage	7
4.1	Type Check Method	7
4.2	Type Validation Method	7
4.3	Type Conversion Methods	8
4.3.1	convert method	8
4.3.2	try_convert method	8
4.3.3	force_convert	8
4.4	For more information	8
5	Installation	9
5.1	Install from PyPI	9
5.2	Install from PPA (for Ubuntu)	9
6	Dependencies	11
6.1	Optioal dependencies	11
6.2	Test dependencies	11
7	Reference	13
7.1	Errors	13
7.2	Type Classes	13
7.2.1	Boolean Type	13
7.2.2	Date Time Type	14
7.2.3	Dictionary Type	16
7.2.4	Infinity Type	17
7.2.5	Integer Type	18
7.2.6	IP address Type	19
7.2.7	List Type	20
7.2.8	NaN Type	21
7.2.9	None Type	22
7.2.10	Null String Type	22
7.2.11	Real Number Type	23

7.2.12 String Type	24
8 Indices and tables	27
9 Links	29
10 Indices and tables	31

CHAPTER 1

typepy

CHAPTER 2

Summary

typepy is a Python library for variable type checker/validator/converter at a run time.

- checking a value type
- validate a value for a type
- convert a value from a type to the other type

The correspondence between Python types and `typepy` classes are as follows:

Table 1: Supported Types

Python Type	<code>typepy</code> : Type Class
<code>bool</code>	<code>Bool</code>
<code>datetime</code>	<code>DateTime</code>
<code>dict</code>	<code>Dictionary</code>
<code>float/decimal.Decimal (not infinity/NaN)</code>	<code>RealNumber</code>
<code>float/decimal.Decimal (infinity)</code>	<code>Infinity</code>
<code>float/decimal.Decimal (NaN)</code>	<code>Nan</code>
<code>int</code>	<code>Integer</code>
<code>list</code>	<code>List</code>
<code>None</code>	<code>None</code>
<code>str (not null)</code>	<code>String</code>
<code>str (null)</code>	<code>NullString</code>
<code>str (IP address)</code>	<code>IpAddress</code>

4.1 Type Check Method

Examples

```
>>> from typepy import Integer
>>> Integer(1).is_type()
True
>>> Integer(1.1).is_type()
False
```

4.2 Type Validation Method

Examples

```
>>> from typepy import Integer
>>> Integer(1).validate()
>>> try:
...     Integer(1.1).validate()
... except TypeError as e:
...     # validate() raised TypeError when the value unmatched the type_
↪class
...     print(e)
...
invalid value type: expected=INTEGER, actual=<type 'float'>
```

4.3 Type Conversion Methods

4.3.1 convert method

Examples

```
>>> from typepy import Integer, TypeConversionError
>>> Integer("1").convert()
1
>>> try:
...     Integer(1.1).convert()
... except TypeConversionError as e:
...     # convert() raised TypeConversionError when conversion failed
...     print(e)
...
failed to convert from float to INTEGER
```

4.3.2 try_convert method

Examples

```
>>> from typepy import Integer
>>> Integer("1").try_convert()
1
>>> print(Integer(1.1).try_convert()) # try_convert() returned None_
↳when conversion failed
None
```

4.3.3 force_convert

Examples

```
>>> from typepy import Integer, TypeConversionError
>>> Integer("1").force_convert() # force_convert() forcibly convert the_
↳value
1
>>> Integer(1.1).force_convert()
1
>>> try:
...     Integer("abc").force_convert()
... except TypeConversionError as e:
...     # force_convert() raised TypeConversionError when the value not_
↳convertible
...     print(e)
...
failed to force_convert to int: type=<class 'str'>
```

4.4 For more information

Type check/validate/convert results differed according to `strict_level` value which can pass to typepy classes constructors as an argument. More information can be found in the [API reference](#).

5.1 Install from PyPI

```
pip install typepy
```

Install additional dependency packages with the following command if using `typepy.DateTime` class

```
pip install typepy[datetime]
```

5.2 Install from PPA (for Ubuntu)

```
sudo add-apt-repository ppa:thombashi/ppa
sudo apt update
sudo apt install python3-typepy
```


Python 2.7+ or 3.5+

- `mbstrdecoder`
- `six`

6.1 Optioal dependencies

These packages can be installed via `pip install typepy[datetime]:`

- `python-dateutil`
- `pytz`

6.2 Test dependencies

- `pytest`
- `pytest-runner`
- `tox`

7.1 Errors

exception `typepy.TypeConversionError`

Bases: `TypeError`

Exception raised when failed to convert data.

7.2 Type Classes

7.2.1 Boolean Type

class `typepy.Bool` (*value*, *strict_level=2*, ***kwargs*)

For each member methods, the result matrix for each `strict_level` is as follows. Column headers (except Method column) indicate input data to `value` argument of a method in the Method column. For each cell shows the output of the method.

Table 1: `typepy.Bool: strict_level = 0`

Method	True	"true"	1
<code>is_type()</code>	True	True	True
<code>validate()</code>	NOP ¹	NOP ¹	NOP ¹
<code>convert()</code>	True	True	True
<code>try_convert()</code>	True	True	True
<code>force_convert()</code>	True	True	True

¹ No Operation

Table 2: typepy.Bool: strict_level = 1

Method	True	"true"	1
is_type()	True	True	False
validate()	NOP ¹	NOP ¹	E ²
convert()	True	True	E ²
try_convert()	True	True	None
force_convert()	True	True	True

Table 3: typepy.Bool: strict_level = 2

Method	True	"true"	1
is_type()	True	False	False
validate()	NOP ¹	E ²	E ²
convert()	True	E ²	E ²
try_convert()	True	None	None
force_convert()	True	True	True

strict_level

Represents how much strict to detect the value type. Higher `strict_level` means that stricter type check.

convert()

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

force_convert()

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

is_type()

Returns

Return type `bool`

try_convert()

Returns Converted value. `None` if failed to convert.

validate (*error_message=None*)

Raises `TypeError` – If the value is not matched the type that the class represented.

7.2.2 Date Time Type

class `typepy.DateTime` (*value, strict_level=2, **kwargs*)

For each member methods, the result matrix for each `strict_level` is as follows. Column headers (except Method column) indicate input data to value argument of a method in the Method column. For each cell shows the output of the method.

² Error (raise `TypeError`)

Table 4: typepy.DateTime: strict_level = 0

Method	datetime(2017, 1, 23, 4, 56)	"2017-01-22T04:56:00+09:00"	1485685623	"1485685623"
is_type()	True	True	True	True
validate()	NOP ¹	NOP ¹	NOP ¹	NOP ¹
convert()	2017-01-23 04:56:00	2017-01-22 04:56:00+09:00	2017-01-29 19:27:03	2017-01-29T19:27:03
try_convert()	2017-01-23 04:56:00	2017-01-22 04:56:00+09:00	2017-01-29 19:27:03	2017-01-29T19:27:03
force_convert()	2017-01-23 04:56:00	2017-01-22 04:56:00+09:00	2017-01-29 19:27:03	2017-01-29T19:27:03

Table 5: typepy.DateTime: strict_level = 1

Method	datetime(2017, 1, 23, 4, 56)	"2017-01-22T04:56:00+09:00"	1485685623	"1485685623"
is_type()	True	True	False	False
validate()	NOP ¹	NOP ¹	E ²	E ²
convert()	2017-01-23 04:56:00	2017-01-22 04:56:00+09:00	E ²	E ²
try_convert()	2017-01-23 04:56:00	2017-01-22 04:56:00+09:00	None	None
force_convert()	2017-01-23 04:56:00	2017-01-22 04:56:00+09:00	2017-01-29 19:27:03	2017-01-29T19:27:03

Table 6: typepy.DateTime: strict_level = 2

Method	datetime(2017, 1, 23, 4, 56)	"2017-01-22T04:56:00+09:00"	1485685623	"1485685623"
is_type()	True	False	False	False
validate()	NOP ¹	E ²	E ²	E ²
convert()	2017-01-23 04:56:00	E ²	E ²	E ²
try_convert()	2017-01-23 04:56:00	None	None	None
force_convert()	2017-01-23 04:56:00	2017-01-22 04:56:00+09:00	2017-01-29 19:27:03	2017-01-29T19:27:03

strict_level Represents how much strict to detect the value type. Higher *strict_level* means that stricter type check.

convert()

Returns Converted value.

Raises *typepy.TypeConversionError* – If the value cannot convert.

force_convert()

Returns Converted value.

Raises *typepy.TypeConversionError* – If the value cannot convert.

is_type()

Returns

Return type bool

`try_convert()`

Returns Converted value. `None` if failed to convert.

`validate(error_message=None)`

Raises `TypeError` – If the value is not matched the type that the class represented.

7.2.3 Dictionary Type

class `typepy.Dictionary(value, strict_level=1, **kwargs)`

For each member methods, the result matrix for each `strict_level` is as follows. Column headers (except Method column) indicate input data to `value` argument of a method in the Method column. For each cell shows the output of the method.

Table 7: `typepy.Dictionary: strict_level = 0`

Method	{}	{"a": 1}	(("a", 1),)
<code>is_type()</code>	True	True	True
<code>validate()</code>	NOP ¹	NOP ¹	NOP ¹
<code>convert()</code>	{}	{'a': 1}	{'a': 1}
<code>try_convert()</code>	{}	{'a': 1}	{'a': 1}
<code>force_convert()</code>	{}	{'a': 1}	{'a': 1}

Table 8: `typepy.Dictionary: strict_level = 1`

Method	{}	{"a": 1}	(("a", 1),)
<code>is_type()</code>	True	True	False
<code>validate()</code>	NOP ¹	NOP ¹	E ²
<code>convert()</code>	{}	{'a': 1}	E ²
<code>try_convert()</code>	{}	{'a': 1}	None
<code>force_convert()</code>	{}	{'a': 1}	{'a': 1}

`strict_level` Represents how much strict to detect the value type. Higher `strict_level` means that stricter type check.

`convert()`

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

`force_convert()`

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

`is_type()`

Returns

Return type `bool`

`try_convert()`

Returns Converted value. `None` if failed to convert.

`validate(error_message=None)`

Raises `TypeError` – If the value is not matched the type that the class represented.

7.2.4 Infinity Type

class `typepy.Infinity` (*value*, *strict_level=1*, ***kwargs*)

For each member methods, the result matrix for each `strict_level` is as follows. Column headers (except Method column) indicate input data to `value` argument of a method in the Method column. For each cell shows the output of the method.

Table 9: `typepy.Infinity: strict_level = 0`

Method	<code>float("inf")</code>	"Infinity"	0.1
<code>is_type()</code>	True	True	False
<code>validate()</code>	NOP ¹	NOP ¹	E ²
<code>convert()</code>	<code>Decimal("inf")</code>	<code>Decimal("inf")</code>	E ²
<code>try_convert()</code>	<code>Decimal("inf")</code>	<code>Decimal("inf")</code>	None
<code>force_convert()</code>	<code>Decimal("inf")</code>	<code>Decimal("inf")</code>	0.1

Table 10: `typepy.Infinity: strict_level = 1`

Method	<code>float("inf")</code>	"Infinity"	0.1
<code>is_type()</code>	True	False	False
<code>validate()</code>	NOP ¹	E ²	E ²
<code>convert()</code>	<code>Decimal("inf")</code>	E ²	E ²
<code>try_convert()</code>	<code>Decimal("inf")</code>	None	None
<code>force_convert()</code>	<code>Decimal("inf")</code>	<code>Decimal("inf")</code>	0.1

strict_level Represents how much strict to detect the value type. Higher `strict_level` means that stricter type check.

convert ()

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

force_convert ()

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

is_type ()

Returns

Return type `bool`

try_convert ()

Returns Converted value. `None` if failed to convert.

validate (*error_message=None*)

Raises `TypeError` – If the value is not matched the type that the class represented.

7.2.5 Integer Type

class typepy.Integer (*value*, *strict_level=1*, ***kwargs*)

For each member methods, the result matrix for each `strict_level` is as follows. Column headers (except Method column) indicate input data to `value` argument of a method in the Method column. For each cell shows the output of the method.

Table 11: typepy.Integer: strict_level = 0

Method	1	1.0	1.1	"1"	"1.0"	"1.1"	True
<code>is_type()</code>	True	True	True	True	True	True	True
<code>validate()</code>	NOP ¹	NOP ¹	NOP ¹	NOP ¹	NOP ¹	NOP ¹	NOP ¹
<code>convert()</code>	1	1	1	1	1	1	1
<code>try_convert()</code>	1	1	1	1	1	1	1
<code>force_convert()</code>	1	1	1	1	1	1	1

Table 12: typepy.Integer: strict_level = 1

Method	1	1.0	1.1	"1"	"1.0"	"1.1"	True
<code>is_type()</code>	True	True	False	True	True	False	False
<code>validate()</code>	NOP ¹	NOP ¹	E ²	NOP ¹	NOP ¹	E ²	E ²
<code>convert()</code>	1	1	E ²	1	1	E ²	E ²
<code>try_convert()</code>	1	1	None	1	1	None	None
<code>force_convert()</code>	1	1	1	1	1	1	1

Table 13: typepy.Integer: strict_level = 2

Method	1	1.0	1.1	"1"	"1.0"	"1.1"	True
<code>is_type()</code>	True	False	False	False	False	False	False
<code>validate()</code>	NOP ¹	E ²	E ²	E ²	E ²	E ²	E ²
<code>convert()</code>	1	E ²	E ²	E ²	E ²	E ²	E ²
<code>try_convert()</code>	1	None	None	None	None	None	None
<code>force_convert()</code>	1	1	1	1	1	1	1

strict_level Represents how much strict to detect the value type. Higher `strict_level` means that stricter type check.

convert()

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

force_convert()

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

is_type()

Returns

Return type `bool`

try_convert()

Returns Converted value. `None` if failed to convert.

validate (*error_message=None*)

Raises **TypeError** – If the value is not matched the type that the class represented.

7.2.6 IP address Type

class typepy.**IpAddress** (*value, strict_level=1, **kwargs*)

For each member methods, the result matrix for each *strict_level* is as follows. Column headers (except Method column) indicate input data to *value* argument of a method in the Method column. For each cell shows the output of the method.

Table 14: typepy.IpAddress: strict_level = 0

Method	ip_address('127.0.0.1')	'127.0.0.1'	'::1'	'192.168.0.256'	None
is_type()	True	True	True	False	False
validate()	NOP ¹	NOP ¹	NOP ¹	E ²	E ²
convert()	ip_address("127.0.0.1")	ip_address("127.0.0.1")	"::1"	E ²	E ²
try_convert()	ip_address("127.0.0.1")	ip_address("127.0.0.1")	"::1"	None	None
force_convert()	ip_address("127.0.0.1")	ip_address("127.0.0.1")	"::1"	E ²	E ²

Table 15: typepy.IpAddress: strict_level = 1

Method	ip_address('127.0.0.1')	'127.0.0.1'	'::1'	'192.168.0.256'	None
is_type()	True	False	False	False	False
validate()	NOP ¹	E ²	E ²	E ²	E ²
convert()	ip_address("127.0.0.1")	E ²	E ²	E ²	E ²
try_convert()	ip_address("127.0.0.1")	None	None	None	None
force_convert()	ip_address("127.0.0.1")	ip_address("127.0.0.1")	"::1"	E ²	E ²

strict_level Represents how much strict to detect the value type. Higher *strict_level* means that stricter type check.

convert ()

Returns Converted value.

Raises **typepy.TypeConversionError** – If the value cannot convert.

force_convert ()

Returns Converted value.

Raises **typepy.TypeConversionError** – If the value cannot convert.

is_type ()

Returns

Return type **bool**

`try_convert()`

Returns Converted value. `None` if failed to convert.

`validate(error_message=None)`

Raises `TypeError` – If the value is not matched the type that the class represented.

7.2.7 List Type

class `typepy.List` (*value*, *strict_level=1*, ***kwargs*)

For each member methods, the result matrix for each `strict_level` is as follows. Column headers (except Method column) indicate input data to `value` argument of a method in the Method column. For each cell shows the output of the method.

Table 16: `typepy.List: strict_level = 0`

Method	[]	["a", "b"]	("a", "b")	{"a": 1}	"abc"
<code>is_type()</code>	True	True	True	True	False
<code>validate()</code>	NOP ¹	NOP ¹	NOP ¹	NOP ¹	E ²
<code>convert()</code>	[]	['a', 'b']	['a', 'b']	['a']	E ²
<code>try_convert()</code>	[]	['a', 'b']	['a', 'b']	['a']	None
<code>force_convert()</code>	[]	['a', 'b']	['a', 'b']	['a']	['a', 'b', 'c']

Table 17: `typepy.List: strict_level = 1`

Method	[]	["a", "b"]	("a", "b")	{"a": 1}	"abc"
<code>is_type()</code>	True	True	False	False	False
<code>validate()</code>	NOP ¹	NOP ¹	E ²	E ²	E ²
<code>convert()</code>	[]	['a', 'b']	E ²	E ²	E ²
<code>try_convert()</code>	[]	['a', 'b']	None	None	None
<code>force_convert()</code>	[]	['a', 'b']	['a', 'b']	['a']	['a', 'b', 'c']

strict_level Represents how much strict to detect the value type. Higher `strict_level` means that stricter type check.

`convert()`

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

`force_convert()`

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

`is_type()`

Returns

Return type `bool`

`try_convert()`

Returns Converted value. `None` if failed to convert.

`validate(error_message=None)`

Raises `TypeError` – If the value is not matched the type that the class represented.

7.2.8 NaN Type

class `typepy.Nan` (*value*, *strict_level=1*, ***kwargs*)

For each member methods, the result matrix for each `strict_level` is as follows. Column headers (except Method column) indicate input data to `value` argument of a method in the Method column. For each cell shows the output of the method.

Table 18: `typepy.Nan: strict_level = 0`

Method	<code>float("nan")</code>	<code>"NaN"</code>	<code>0.1</code>
<code>is_type()</code>	True	True	False
<code>validate()</code>	NOP ¹	NOP ¹	E ²
<code>convert()</code>	<code>Decimal("nan")</code>	<code>Decimal("nan")</code>	E ²
<code>try_convert()</code>	<code>Decimal("nan")</code>	<code>Decimal("nan")</code>	None
<code>force_convert()</code>	<code>Decimal("nan")</code>	<code>Decimal("nan")</code>	<code>0.1</code>

Table 19: `typepy.Nan: strict_level = 1`

Method	<code>float("nan")</code>	<code>"NaN"</code>	<code>0.1</code>
<code>is_type()</code>	True	False	False
<code>validate()</code>	NOP ¹	E ²	E ²
<code>convert()</code>	<code>Decimal("nan")</code>	E ²	E ²
<code>try_convert()</code>	<code>Decimal("nan")</code>	None	None
<code>force_convert()</code>	<code>Decimal("nan")</code>	<code>Decimal("nan")</code>	<code>0.1</code>

strict_level Represents how much strict to detect the value type. Higher `strict_level` means that stricter type check.

convert ()

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

force_convert ()

Returns Converted value.

Raises `typepy.TypeConversionError` – If the value cannot convert.

is_type ()

Returns

Return type `bool`

try_convert ()

Returns Converted value. `None` if failed to convert.

validate (*error_message=None*)

Raises `TypeError` – If the value is not matched the type that the class represented.

7.2.9 None Type

class typepy.**NoneType** (*value*, *strict_level=0*, ***kwargs*)

For each member methods, the result matrix for each *strict_level* is as follows. Column headers (except Method column) indicate input data to *value* argument of a method in the Method column. For each cell shows the output of the method.

Table 20: typepy.NoneType: *strict_level* = 0

Method	"abc"	""	" "	None	1
<code>is_type()</code>	False	False	False	True	False
<code>validate()</code>	E ²	E ²	E ²	NOP ¹	E ²
<code>convert()</code>	E ²	E ²	E ²	None	E ²
<code>try_convert()</code>	None	None	None	None	None
<code>force_convert()</code>	None	None	None	None	None

strict_level Represents how much strict to detect the value type. Higher *strict_level* means that stricter type check.

convert ()

Returns Converted value.

Raises *typepy.TypeConversionError* – If the value cannot convert.

force_convert ()

Returns Converted value.

Raises *typepy.TypeConversionError* – If the value cannot convert.

is_type ()

Returns

Return type bool

try_convert ()

Returns Converted value. None if failed to convert.

validate (*error_message=None*)

Raises *TypeError* – If the value is not matched the type that the class represented.

7.2.10 Null String Type

class typepy.**NullString** (*value*, *strict_level=1*, ***kwargs*)

For each member methods, the result matrix for each *strict_level* is as follows. Column headers (except Method column) indicate input data to *value* argument of a method in the Method column. For each cell shows the output of the method.

Table 21: typepy.NullString: *strict_level* = 0

Method	"abc"	""	" "	None	1
<code>is_type()</code>	False	True	True	True	False
<code>validate()</code>	E ²	NOP ¹	NOP ¹	NOP ¹	E ²
<code>convert()</code>	E ²	""	""	""	E ²
<code>try_convert()</code>	None	""	""	""	None
<code>force_convert()</code>	""	""	""	""	""

Table 22: typepy.NullString: strict_level = 1

Method	"abc"	""	" "	None	1
is_type()	False	True	True	False	False
validate()	E ²	NOP ¹	NOP ¹	E ²	E ²
convert()	E ²	""	""	E ²	E ²
try_convert()	None	""	""	None	None
force_convert()	""	""	""	""	""

strict_level Represents how much strict to detect the value type. Higher *strict_level* means that stricter type check.

convert()

Returns Converted value.

Raises *typepy.TypeConversionError* – If the value cannot convert.

force_convert()

Returns Converted value.

Raises *typepy.TypeConversionError* – If the value cannot convert.

is_type()

Returns

Return type bool

try_convert()

Returns Converted value. None if failed to convert.

validate (*error_message=None*)

Raises *TypeError* – If the value is not matched the type that the class represented.

7.2.11 Real Number Type

class typepy.RealNumber (*value, strict_level=0, **kwargs*)

For each member methods, the result matrix for each *strict_level* is as follows. Column headers (except Method column) indicate input data to *value* argument of a method in the Method column. For each cell shows the output of the method.

Table 23: typepy.RealNumber: strict_level = 0

Method	1	1.0	1.1	"1"	"1.0"	"1.1"	True
is_type()	True	True	True	True	True	True	False
validate()	NOP ¹	NOP ¹	NOP ¹	NOP ¹	NOP ¹	NOP ¹	E ²
convert()	1	1	1.1	1	1	1.1	E ²
try_convert()	1	1	1.1	1	1	1.1	None
force_convert()	1	1	1.1	1	1	1.1	1

Table 24: typepy.RealNumber: strict_level = 1

Method	1	1.0	1.1	"1"	"1.0"	"1.1"	True
is_type ()	False	False	True	False	False	True	False
validate ()	E ²	E ²	NOP ¹	E ²	E ²	NOP ¹	E ²
convert ()	E ²	E ²	1.1	E ²	E ²	1.1	E ²
try_convert ()	None	None	1.1	None	None	1.1	None
force_convert ()	1	1	1.1	1	1	1.1	1

Table 25: typepy.RealNumber: strict_level = 2

Method	1	1.0	1.1	"1"	"1.0"	"1.1"	True
is_type ()	False	False	True	False	False	False	False
validate ()	E ²	E ²	NOP ¹	E ²	E ²	E ²	E ²
convert ()	E ²	E ²	1.1	E ²	E ²	E ²	E ²
try_convert ()	None	None	1.1	None	None	None	None
force_convert ()	1	1	1.1	1	1	1.1	1

strict_level Represents how much strict to detect the value type. Higher *strict_level* means that stricter type check.

convert ()

Returns Converted value.

Raises *typepy.TypeConversionError* – If the value cannot convert.

force_convert ()

Returns Converted value.

Raises *typepy.TypeConversionError* – If the value cannot convert.

is_type ()

Returns

Return type `bool`

try_convert ()

Returns Converted value. `None` if failed to convert.

validate (error_message=None)

Raises *TypeError* – If the value is not matched the type that the class represented.

7.2.12 String Type

class typepy.String (value, strict_level=1, **kwargs)

For each member methods, the result matrix for each *strict_level* is as follows. Column headers (except Method column) indicate input data to value argument of a method in the Method column. For each cell shows the output of the method.

Table 26: typepy.String: strict_level = 0

Method	"abc"	""	" "	None	1
is_type()	True	True	True	True	True
validate()	NOP ¹	NOP ¹	NOP ¹	NOP ¹	NOP ¹
convert()	"abc"	""	" "	"None"	"1"
try_convert()	"abc"	""	" "	"None"	"1"
force_convert()	"abc"	""	" "	"None"	"1"

Table 27: typepy.String: strict_level = 1

Method	"abc"	""	" "	None	1
is_type()	True	True	True	False	False
validate()	NOP ¹	NOP ¹	NOP ¹	E ²	E ²
convert()	"abc"	""	" "	E ²	E ²
try_convert()	"abc"	""	" "	None	None
force_convert()	"abc"	""	" "	"None"	"1"

Table 28: typepy.String: strict_level = 2

Method	"abc"	""	" "	None	1
is_type()	True	False	False	False	False
validate()	NOP ¹	E ²	E ²	E ²	E ²
convert()	"abc"	E ²	E ²	E ²	E ²
try_convert()	"abc"	None	None	None	None
force_convert()	"abc"	""	" "	"None"	"1"

strict_level Represents how much strict to detect the value type. Higher *strict_level* means that stricter type check.

convert()

Returns Converted value.

Raises *typepy.TypeConversionError* – If the value cannot convert.

force_convert()

Returns Converted value.

Raises *typepy.TypeConversionError* – If the value cannot convert.

is_type()

Returns

Return type bool

try_convert()

Returns Converted value. *None* if failed to convert.

validate (*error_message=None*)

Raises *TypeError* – If the value is not matched the type that the class represented.

CHAPTER 8

Indices and tables

- `genindex`

CHAPTER 9

Links

- [GitHub repository](#)
- [Issue tracker](#)
- [PyPI page](#)
- [pip: tool for installing Python packages](#)

CHAPTER 10

Indices and tables

- `genindex`

B

Bool (class in typepy), 13

C

convert () (typepy.Bool method), 14
 convert () (typepy.DateTime method), 15
 convert () (typepy.Dictionary method), 16
 convert () (typepy.Infinity method), 17
 convert () (typepy.Integer method), 18
 convert () (typepy.IpAddress method), 19
 convert () (typepy.List method), 20
 convert () (typepy.Nan method), 21
 convert () (typepy.NoneType method), 22
 convert () (typepy.NullString method), 23
 convert () (typepy.RealNumber method), 24
 convert () (typepy.String method), 25

D

DateTime (class in typepy), 14
 Dictionary (class in typepy), 16

F

force_convert () (typepy.Bool method), 14
 force_convert () (typepy.DateTime method), 15
 force_convert () (typepy.Dictionary method), 16
 force_convert () (typepy.Infinity method), 17
 force_convert () (typepy.Integer method), 18
 force_convert () (typepy.IpAddress method), 19
 force_convert () (typepy.List method), 20
 force_convert () (typepy.Nan method), 21
 force_convert () (typepy.NoneType method), 22
 force_convert () (typepy.NullString method), 23
 force_convert () (typepy.RealNumber method), 24
 force_convert () (typepy.String method), 25

I

Infinity (class in typepy), 17
 Integer (class in typepy), 18
 IpAddress (class in typepy), 19

is_type () (typepy.Bool method), 14
 is_type () (typepy.DateTime method), 15
 is_type () (typepy.Dictionary method), 16
 is_type () (typepy.Infinity method), 17
 is_type () (typepy.Integer method), 18
 is_type () (typepy.IpAddress method), 19
 is_type () (typepy.List method), 20
 is_type () (typepy.Nan method), 21
 is_type () (typepy.NoneType method), 22
 is_type () (typepy.NullString method), 23
 is_type () (typepy.RealNumber method), 24
 is_type () (typepy.String method), 25

L

List (class in typepy), 20

N

Nan (class in typepy), 21
 NoneType (class in typepy), 22
 NullString (class in typepy), 22

R

RealNumber (class in typepy), 23

S

strict_level (typepy.Bool attribute), 14
 String (class in typepy), 24

T

try_convert () (typepy.Bool method), 14
 try_convert () (typepy.DateTime method), 15
 try_convert () (typepy.Dictionary method), 16
 try_convert () (typepy.Infinity method), 17
 try_convert () (typepy.Integer method), 18
 try_convert () (typepy.IpAddress method), 19
 try_convert () (typepy.List method), 20
 try_convert () (typepy.Nan method), 21
 try_convert () (typepy.NoneType method), 22
 try_convert () (typepy.NullString method), 23

`try_convert()` (*typepy.RealNumber method*), 24
`try_convert()` (*typepy.String method*), 25
`TypeConversionError`, 13

V

`validate()` (*typepy.Bool method*), 14
`validate()` (*typepy.DateTime method*), 16
`validate()` (*typepy.Dictionary method*), 16
`validate()` (*typepy.Infinity method*), 17
`validate()` (*typepy.Integer method*), 18
`validate()` (*typepy.IpAddress method*), 20
`validate()` (*typepy.List method*), 20
`validate()` (*typepy.Nan method*), 21
`validate()` (*typepy.NoneType method*), 22
`validate()` (*typepy.NullString method*), 23
`validate()` (*typepy.RealNumber method*), 24
`validate()` (*typepy.String method*), 25