typepy Documentation

Release 1.3.0

Tsuyoshi Hombashi

Mar 27, 2022

TABLE OF CONTENTS

CHAPTER

ONE

TYPEPY

CHAPTER

TWO

SUMMARY

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

CHAPTER

THREE

FEATURES

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

CHAPTER

FOUR

INSTALLATION

4.1 Installation: pip

```
pip install typepy
```

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

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

4.2 Installation: conda

```
conda install -c conda-forge typepy
```

4.3 Installation: apt

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

CHAPTER

FIVE

DEPENDENCIES

- Python 3.6+
- Python package dependencies (automatically installed)

5.1 Optioal dependencies

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

- python-dateutil
- pytz

CHAPTER

SIX

USAGE

6.1 Type Check Method

Examples

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

6.2 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'>
```

6.3 Type Conversion Methods

6.3.1 convert method

Examples

```
>>> from typepy import Integer, TypeConversionError
>>> Integer("1").convert()
1
>>> try:
```

(continues on next page)

(continued from previous page)

```
...     Integer(1.1).convert()
... except TypeConversionError as e:
...     # convert() raised TypeConversionError when conversion failed
...     print(e)
...
failed to convert from float to INTEGER
```

6.3.2 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
```

6.3.3 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'>
```

6.4 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.

CHAPTER

SEVEN

REFERENCE

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: Any*, *strict_level: int = 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
is_type()	True	True	True
validate()	NOP[1]	NOP[?]	NOP[?]
convert()	True	True	True
try_convert()	True	True	True
force_convert()	True	True	True

Table 2: `typepy.Bool`: `strict_level` = 1

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

[1] No Operation
[2] Error (raise TypeError)

Table 3: `typepy.Bool`: `strict_level` = 2

Method	`True`	`"true"`	`1`
`is_type()`	`True`	`False`	`False`
`validate()`	NOP[?]	E[?]	E[?]
`convert()`	`True`	E[?]	E[?]
`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() → bool

Returns

Return type bool

try_convert()

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

validate(*error_message: Optional[str] = None*) → 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: Any*, *strict_level: int = 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 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[?]	NOP[?]	NOP[?]	NOP[?]
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[?]	NOP[?]	E[?]	E[?]
convert()	2017-01-23 04:56:00	2017-01-22 04:56:00+09:00	E[?]	E[?]
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[?]	E[?]	E[?]	E[?]
convert()	2017-01-23 04:56:00	E[?]	E[?]	E[?]
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() → bool

Returns

Return type bool

`try_convert()`

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

`validate`(*error_message: Optional[str] = None*) → None

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

7.2.3 Dictionary Type

`class typepy.Dictionary`(*value: Any*, *strict_level: int = 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),)
is_type()	True	True	True
validate()	NOP[?]	NOP[?]	NOP[?]
convert()	{}	{'a': 1}	{'a': 1}
try_convert()	{}	{'a': 1}	{'a': 1}
force_convert()	{}	{'a': 1}	{'a': 1}

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

Method	{}	{"a": 1}	(("a", 1),)
is_type()	True	True	False
validate()	NOP[?]	NOP[?]	E[?]
convert()	{}	{'a': 1}	E[?]
try_convert()	{}	{'a': 1}	None
force_convert()	{}	{'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()` → bool

Returns

Return type bool

`try_convert()`

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

`validate(error_message: Optional[str] = None) → None`

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

7.2.4 Infinity Type

`class typepy.Infinity(value: Any, strict_level: int = 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	`float("inf")`	`"Infinity"`	`0.1`
`is_type()`	`True`	`True`	`False`
`validate()`	NOP[?]	NOP[?]	E[?]
`convert()`	`Decimal("inf")`	`Decimal("inf")`	E[?]
`try_convert()`	`Decimal("inf")`	`Decimal("inf")`	`None`
`force_convert()`	`Decimal("inf")`	`Decimal("inf")`	0.1

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

Method	`float("inf")`	`"Infinity"`	`0.1`
`is_type()`	`True`	`False`	`False`
`validate()`	NOP[?]	E[?]	E[?]
`convert()`	`Decimal("inf")`	E[?]	E[?]
`try_convert()`	`Decimal("inf")`	`None`	`None`
`force_convert()`	`Decimal("inf")`	`Decimal("inf")`	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() → bool`

Returns

Return type bool

`try_convert()`

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

`validate`(*error_message: Optional[str] = None*) → None

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

7.2.5 Integer Type

`class typepy.Integer`(*value: Any*, *strict_level: int = 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
is_type()	True	True	True	True	True	True	True
validate()	NOP[?]	NOP[?]	NOP[?]	NOP[?]	NOP[?]	NOP[?]	NOP[?]
convert()	1	1	1	1	1	1	1
try_convert()	1	1	1	1	1	1	1
force_convert()	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
is_type()	True	True	False	True	True	False	False
validate()	NOP[?]	NOP[?]	E[?]	NOP[?]	NOP[?]	E[?]	E[?]
convert()	1	1	E[?]	1	1	E[?]	E[?]
try_convert()	1	1	None	1	1	None	None
force_convert()	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
is_type()	True	False	False	False	False	False	False
validate()	NOP[?]	E[?]	E[?]	E[?]	E[?]	E[?]	E[?]
convert()	1	E[?]	E[?]	E[?]	E[?]	E[?]	E[?]
try_convert()	1	None	None	None	None	None	None
force_convert()	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`() → bool

Returns

Return type bool

`try_convert`()

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

`validate`(*error_message: Optional[str] = None*) → 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: Any*, *strict_level: int = 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[?]	NOP[?]	NOP[?]	E[?]	E[?]
`convert()`	`ip_address("127.0.0.1")`	`ip_address("127.0.0.1")`	`"::1"`	E[?]	E[?]
`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[?]	E[?]

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[?]	E[?]	E[?]	E[?]	E[?]
`convert()`	`ip_address("127.0.0.1")`	E[?]	E[?]	E[?]	E[?]
`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[?]	E[?]

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()` → bool

Returns

Return type bool

`try_convert()`

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

`validate`(*error_message: Optional[str] = None*) → None

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

7.2.7 List Type

class typepy.**List**(*value: Any*, *strict_level: int = 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"
is_type()	True	True	True	True	False
validate()	NOP[?]	NOP[?]	NOP[?]	NOP[?]	E[?]
convert()	[]	['a', 'b']	['a', 'b']	['a']	E[?]
try_convert()	[]	['a', 'b']	['a', 'b']	['a']	None
force_convert()	[]	['a', 'b']	['a', 'b']	['a']	['a', 'b', 'c']

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

Method	[]	["a", "b"]	("a", "b")	{"a": 1}	"abc"
is_type()	True	True	False	False	False
validate()	NOP[?]	NOP[?]	E[?]	E[?]	E[?]
convert()	[]	['a', 'b']	E[?]	E[?]	E[?]
try_convert()	[]	['a', 'b']	None	None	None
force_convert()	[]	['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()` → bool

Returns

Return type bool

`try_convert()`

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

`validate`(*error_message: Optional[str] = None*) → None

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

7.2.8 NaN Type

class typepy.**Nan**(*value: Any*, *strict_level: int = 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	`float("nan")`	`"NaN"`	`0.1`
`is_type()`	`True`	`True`	`False`
`validate()`	NOP[?]	NOP[?]	E[?]
`convert()`	`Decimal("nan")`	`Decimal("nan")`	E[?]
`try_convert()`	`Decimal("nan")`	`Decimal("nan")`	`None`
`force_convert()`	`Decimal("nan")`	`Decimal("nan")`	0.1

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

Method	`float("nan")`	`"NaN"`	`0.1`
`is_type()`	`True`	`False`	`False`
`validate()`	NOP[?]	E[?]	E[?]
`convert()`	`Decimal("nan")`	E[?]	E[?]
`try_convert()`	`Decimal("nan")`	`None`	`None`
`force_convert()`	`Decimal("nan")`	`Decimal("nan")`	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() → bool

Returns

Return type bool

try_convert()

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

validate(*error_message: Optional[str] = None*) → None

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

7.2.9 None Type

class typepy.**NoneType**(*value: Any*, *strict_level: int = 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
is_type()	False	False	False	True	False
validate()	E[?]	E[?]	E[?]	NOP[?]	E[?]
convert()	E[?]	E[?]	E[?]	None	E[?]
try_convert()	None	None	None	None	None
force_convert()	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() → bool

Returns

Return type bool

try_convert()

Returns Converted value. None if failed to convert.

validate(*error_message: Optional[str] = None*) → 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: Any*, *strict_level: int = 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
is_type()	False	True	True	True	False
validate()	E[?]	NOP[?]	NOP[?]	NOP[?]	E[?]
convert()	E[?]	""	""	""	E[?]
try_convert()	None	""	""	""	None
force_convert()	""	""	""	""	""

Table 22: typepy.NullString: strict_level = 1

Method	"abc"	""	" "	None	1
is_type()	False	True	True	False	False
validate()	E[?]	NOP[?]	NOP[?]	E[?]	E[?]
convert()	E[?]	""	""	E[?]	E[?]
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`() → bool

Returns

Return type bool

`try_convert`()

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

`validate`(*error_message: Optional[str] = None*) → 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: Any*, *strict_level: int = 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[?]	NOP[?]	NOP[?]	NOP[?]	NOP[?]	NOP[?]	E[?]
convert()	1	1	1.1	1	1	1.1	E[?]
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[?]	E[?]	NOP[?]	E[?]	E[?]	NOP[?]	E[?]
convert()	E[?]	E[?]	1.1	E[?]	E[?]	1.1	E[?]
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[?]	E[?]	NOP[?]	E[?]	E[?]	E[?]	E[?]
convert()	E[?]	E[?]	1.1	E[?]	E[?]	E[?]	E[?]
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()` → bool

Returns

Return type bool

`try_convert()`

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

`validate`(*error_message: Optional[str] = None*) → None

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

7.2.12 String Type

class typepy.**String**(*value: Any*, *strict_level: int = 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[?]	NOP[?]	NOP[?]	NOP[?]	NOP[?]
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[?]	NOP[?]	NOP[?]	E[?]	E[?]
convert()	"abc"	""	" "	E[?]	E[?]
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[?]	E[?]	E[?]	E[?]	E[?]
`convert()`	`"abc"`	E[?]	E[?]	E[?]	E[?]
`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() → bool

Returns

Return type bool

try_convert()

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

validate(*error_message: Optional[str] = None*) → None

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

CHAPTER

EIGHT

CHANGELOG

https://github.com/thombashi/typepy/releases

CHAPTER

NINE

INDICES AND TABLES

- genindex

CHAPTER

TEN

LINKS

- GitHub repository
- Issue tracker
- pip: tool for installing Python packages

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex

INDEX

V