

---

# **typepy Documentation**

*Release 1.3.0*

**Tsuyoshi Hombashi**

**Mar 27, 2022**



# TABLE OF CONTENTS

|          |                                   |           |
|----------|-----------------------------------|-----------|
| <b>1</b> | <b>typepy</b>                     | <b>1</b>  |
| <b>2</b> | <b>Summary</b>                    | <b>3</b>  |
| <b>3</b> | <b>Features</b>                   | <b>5</b>  |
| <b>4</b> | <b>Installation</b>               | <b>7</b>  |
| 4.1      | Installation: pip . . . . .       | 7         |
| 4.2      | Installation: conda . . . . .     | 7         |
| 4.3      | Installation: apt . . . . .       | 7         |
| <b>5</b> | <b>Dependencies</b>               | <b>9</b>  |
| 5.1      | Optioal dependencies . . . . .    | 9         |
| <b>6</b> | <b>Usage</b>                      | <b>11</b> |
| 6.1      | Type Check Method . . . . .       | 11        |
| 6.2      | Type Validation Method . . . . .  | 11        |
| 6.3      | Type Conversion Methods . . . . . | 11        |
| 6.3.1    | convert method . . . . .          | 11        |
| 6.3.2    | try_convert method . . . . .      | 12        |
| 6.3.3    | force_convert . . . . .           | 12        |
| 6.4      | For more information . . . . .    | 12        |
| <b>7</b> | <b>Reference</b>                  | <b>13</b> |
| 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 . . . . .        | 23        |
| 7.2.11   | Real Number Type . . . . .        | 24        |
| 7.2.12   | String Type . . . . .             | 25        |
| <b>8</b> | <b>Changelog</b>                  | <b>27</b> |

|           |                           |           |
|-----------|---------------------------|-----------|
| <b>9</b>  | <b>Indices and tables</b> | <b>29</b> |
| <b>10</b> | <b>Links</b>              | <b>31</b> |
| <b>11</b> | <b>Indices and tables</b> | <b>33</b> |
|           | <b>Index</b>              | <b>35</b> |

---

CHAPTER  
**ONE**

---

**TYPEPY**



## SUMMARY

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





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



## 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
```



## 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`



## 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](#).



## 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                |
|------------------------------|------------------|------------------|------------------|
| <code>is_type()</code>       | True             | True             | True             |
| <code>validate()</code>      | NOP <sup>1</sup> | NOP <sup>2</sup> | NOP <sup>2</sup> |
| <code>convert()</code>       | True             | True             | True             |
| <code>try_convert()</code>   | True             | True             | True             |
| <code>force_convert()</code> | True             | True             | True             |

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

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

---

<sup>1</sup> No Operation

<sup>2</sup> 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), ))   |
|------------------------------|------------------|------------------|------------------|
| <code>is_type()</code>       | True             | True             | True             |
| <code>validate()</code>      | NOP <sup>?</sup> | NOP <sup>?</sup> | NOP <sup>?</sup> |
| <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 <sup>?</sup> | NOP <sup>?</sup> | E <sup>?</sup> |
| <code>convert()</code>       | {}               | {'a': 1}         | E <sup>?</sup> |
| <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() → 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                       | <code>float("inf")</code>   | <code>"Infinity"</code>     | <code>0.1</code> |
|------------------------------|-----------------------------|-----------------------------|------------------|
| <code>is_type()</code>       | True                        | True                        | False            |
| <code>validate()</code>      | NOP <sup>?</sup>            | NOP <sup>?</sup>            | E <sup>?</sup>   |
| <code>convert()</code>       | <code>Decimal("inf")</code> | <code>Decimal("inf")</code> | E <sup>?</sup>   |
| <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>   | <code>"Infinity"</code>     | <code>0.1</code> |
|------------------------------|-----------------------------|-----------------------------|------------------|
| <code>is_type()</code>       | True                        | False                       | False            |
| <code>validate()</code>      | NOP <sup>?</sup>            | E <sup>?</sup>              | E <sup>?</sup>   |
| <code>convert()</code>       | <code>Decimal("inf")</code> | E <sup>?</sup>              | E <sup>?</sup>   |
| <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() → 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 |
|------------------------------|------|------|------|------|-------|-------|------|
| <code>is_type()</code>       | True | True | True | True | True  | True  | True |
| <code>validate()</code>      | NOP? | NOP? | NOP? | NOP? | NOP?  | NOP?  | NOP? |
| <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? | NOP? | E?    | NOP? | NOP?  | E?    | E?    |
| <code>convert()</code>       | 1    | 1    | E?    | 1    | 1     | E?    | E?    |
| <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? | E?    | E?    | E?    | E?    | E?    | E?    |
| <code>convert()</code>       | 1    | E?    | E?    | E?    | E?    | E?    | E?    |
| <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()` → 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                       | <code>ip_address('127.0.0.1')</code> | <code>'127.0.0.1'</code>             | <code>:::1'</code> | <code>'192.168.0.256'</code> | None           |
|------------------------------|--------------------------------------|--------------------------------------|--------------------|------------------------------|----------------|
| <code>is_type()</code>       | True                                 | True                                 | True               | False                        | False          |
| <code>validate()</code>      | NOP <sup>?</sup>                     | NOP <sup>?</sup>                     | NOP <sup>?</sup>   | E <sup>?</sup>               | E <sup>?</sup> |
| <code>convert()</code>       | <code>ip_address("127.0.0.1")</code> | <code>ip_address("127.0.0.1")</code> | <code>:::1"</code> | E <sup>?</sup>               | E <sup>?</sup> |
| <code>try_convert()</code>   | <code>ip_address("127.0.0.1")</code> | <code>ip_address("127.0.0.1")</code> | <code>:::1"</code> | None                         | None           |
| <code>force_convert()</code> | <code>ip_address("127.0.0.1")</code> | <code>ip_address("127.0.0.1")</code> | <code>:::1"</code> | E <sup>?</sup>               | E <sup>?</sup> |

Table 15: `typepy.IPAddress: strict_level = 1`

| Method                       | <code>ip_address('127.0.0.1')</code> | <code>'127.0.0.1'</code>             | <code>:::1'</code> | <code>'192.168.0.256'</code> | None           |
|------------------------------|--------------------------------------|--------------------------------------|--------------------|------------------------------|----------------|
| <code>is_type()</code>       | True                                 | False                                | False              | False                        | False          |
| <code>validate()</code>      | NOP <sup>?</sup>                     | E <sup>?</sup>                       | E <sup>?</sup>     | E <sup>?</sup>               | E <sup>?</sup> |
| <code>convert()</code>       | <code>ip_address("127.0.0.1")</code> | E <sup>?</sup>                       | E <sup>?</sup>     | E <sup>?</sup>               | E <sup>?</sup> |
| <code>try_convert()</code>   | <code>ip_address("127.0.0.1")</code> | None                                 | None               | None                         | None           |
| <code>force_convert()</code> | <code>ip_address("127.0.0.1")</code> | <code>ip_address("127.0.0.1")</code> | <code>:::1"</code> | E <sup>?</sup>               | E <sup>?</sup> |

**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"           |
|------------------------------|------|------------|------------|----------|-----------------|
| <code>is_type()</code>       | True | True       | True       | True     | False           |
| <code>validate()</code>      | NOP? | NOP?       | NOP?       | NOP?     | E?              |
| <code>convert()</code>       | []   | ['a', 'b'] | ['a', 'b'] | ['a']    | E?              |
| <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? | NOP?       | E?         | E?       | E?              |
| <code>convert()</code>       | []   | ['a', 'b'] | E?         | E?       | E?              |
| <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()** → 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                       | <code>float("nan")</code>   | <code>"NaN"</code>          | <code>0.1</code> |
|------------------------------|-----------------------------|-----------------------------|------------------|
| <code>is_type()</code>       | True                        | True                        | False            |
| <code>validate()</code>      | NOP <sup>?</sup>            | NOP <sup>?</sup>            | E <sup>?</sup>   |
| <code>convert()</code>       | <code>Decimal("nan")</code> | <code>Decimal("nan")</code> | E <sup>?</sup>   |
| <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> | 0.1              |

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 <sup>?</sup>            | E <sup>?</sup>              | E <sup>?</sup>   |
| <code>convert()</code>       | <code>Decimal("nan")</code> | E <sup>?</sup>              | E <sup>?</sup>   |
| <code>try_convert()</code>   | <code>Decimal("nan")</code> | None                        | None             |
| <code>force_convert()</code> | <code>Decimal("nan")</code> | <code>Decimal("nan")</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()** → 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     |
|------------------------------|-------|-------|-------|------|-------|
| <code>is_type()</code>       | False | False | False | True | False |
| <code>validate()</code>      | E?    | E?    | E?    | NOP? | E?    |
| <code>convert()</code>       | E?    | E?    | E?    | None | E?    |
| <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()` → 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     |
|------------------------------|-------|------|------|------|-------|
| <code>is_type()</code>       | False | True | True | True | False |
| <code>validate()</code>      | E?    | NOP? | NOP? | NOP? | E?    |
| <code>convert()</code>       | E?    | ""   | ""   | ""   | E?    |
| <code>try_convert()</code>   | None  | ""   | ""   | ""   | None  |
| <code>force_convert()</code> | ""    | ""   | ""   | ""   | ""    |

Table 22: `typepy.NullString: strict_level = 1`

| Method                       | "abc" | ""   | " "  | None  | 1     |
|------------------------------|-------|------|------|-------|-------|
| <code>is_type()</code>       | False | True | True | False | False |
| <code>validate()</code>      | E?    | NOP? | NOP? | E?    | E?    |
| <code>convert()</code>       | E?    | ""   | ""   | E?    | E?    |
| <code>try_convert()</code>   | None  | ""   | ""   | None  | None  |
| <code>force_convert()</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()` → 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  |
|------------------------------|------|------|------|------|-------|-------|-------|
| <code>is_type()</code>       | True | True | True | True | True  | True  | False |
| <code>validate()</code>      | NOP? | NOP? | NOP? | NOP? | NOP?  | NOP?  | E?    |
| <code>convert()</code>       | 1    | 1    | 1.1  | 1    | 1     | 1.1   | E?    |
| <code>try_convert()</code>   | 1    | 1    | 1.1  | 1    | 1     | 1.1   | None  |
| <code>force_convert()</code> | 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  |
|------------------------------|-------|-------|------|-------|-------|-------|-------|
| <code>is_type()</code>       | False | False | True | False | False | True  | False |
| <code>validate()</code>      | E?    | E?    | NOP? | E?    | E?    | NOP?  | E?    |
| <code>convert()</code>       | E?    | E?    | 1.1  | E?    | E?    | 1.1   | E?    |
| <code>try_convert()</code>   | None  | None  | 1.1  | None  | None  | 1.1   | None  |
| <code>force_convert()</code> | 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  |
|------------------------------|-------|-------|------|-------|-------|-------|-------|
| <code>is_type()</code>       | False | False | True | False | False | False | False |
| <code>validate()</code>      | E?    | E?    | NOP? | E?    | E?    | E?    | E?    |
| <code>convert()</code>       | E?    | E?    | 1.1  | E?    | E?    | E?    | E?    |
| <code>try_convert()</code>   | None  | None  | 1.1  | None  | None  | None  | None  |
| <code>force_convert()</code> | 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    |
|------------------------------|-------|------|------|--------|------|
| <code>is_type()</code>       | True  | True | True | True   | True |
| <code>validate()</code>      | NOP?  | NOP? | NOP? | NOP?   | NOP? |
| <code>convert()</code>       | "abc" | ""   | " "  | "None" | "1"  |
| <code>try_convert()</code>   | "abc" | ""   | " "  | "None" | "1"  |
| <code>force_convert()</code> | "abc" | ""   | " "  | "None" | "1"  |

Table 27: `typepy.String: strict_level = 1`

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

Table 28: typepy.String: strict\_level = 2

| Method          | "abc"            | ""             | " "            | None           | 1              |
|-----------------|------------------|----------------|----------------|----------------|----------------|
| is_type()       | True             | False          | False          | False          | False          |
| validate()      | NOP <sup>?</sup> | E <sup>?</sup> | E <sup>?</sup> | E <sup>?</sup> | E <sup>?</sup> |
| convert()       | "abc"            | E <sup>?</sup> | E <sup>?</sup> | E <sup>?</sup> | E <sup>?</sup> |
| 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.

**CHANGELOG**

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





## INDICES AND TABLES

- `genindex`



**LINKS**

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



## 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), 20  
 convert() (typepy.List method), 21  
 convert() (typepy.Nan method), 22  
 convert() (typepy.NoneType method), 22  
 convert() (typepy.NullString method), 23  
 convert() (typepy.RealNumber method), 25  
 convert() (typepy.String method), 26

## 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), 19  
 force\_convert() (typepy.IpAddress method), 20  
 force\_convert() (typepy.List method), 21  
 force\_convert() (typepy.Nan method), 22  
 force\_convert() (typepy.NoneType method), 22  
 force\_convert() (typepy.NullString method), 23  
 force\_convert() (typepy.RealNumber method), 25  
 force\_convert() (typepy.String method), 26

## 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), 19  
 is\_type() (typepy.IpAddress method), 20  
 is\_type() (typepy.List method), 21  
 is\_type() (typepy.Nan method), 22  
 is\_type() (typepy.NoneType method), 23  
 is\_type() (typepy.NullString method), 24  
 is\_type() (typepy.RealNumber method), 25  
 is\_type() (typepy.String method), 26

## L

List (class in typepy), 20

## N

Nan (class in typepy), 21  
 NoneType (class in typepy), 22  
 NullString (class in typepy), 23

## R

RealNumber (class in typepy), 24

## S

strict\_level (typepy.Bool attribute), 14  
 String (class in typepy), 25

## T

try\_convert() (typepy.Bool method), 14  
 try\_convert() (typepy.DateTime method), 16  
 try\_convert() (typepy.Dictionary method), 17  
 try\_convert() (typepy.Infinity method), 18  
 try\_convert() (typepy.Integer method), 19  
 try\_convert() (typepy.IpAddress method), 20  
 try\_convert() (typepy.List method), 21  
 try\_convert() (typepy.Nan method), 22  
 try\_convert() (typepy.NoneType method), 23  
 try\_convert() (typepy.NullString method), 24  
 try\_convert() (typepy.RealNumber method), 25  
 try\_convert() (typepy.String method), 26  
 TypeConversionError, 13

## V

- `validate()` (*typepy.Bool method*), 14
- `validate()` (*typepy.DateTime method*), 16
- `validate()` (*typepy.Dictionary method*), 17
- `validate()` (*typepy.Infinity method*), 18
- `validate()` (*typepy.Integer method*), 19
- `validate()` (*typepy.IpAddress method*), 20
- `validate()` (*typepy.List method*), 21
- `validate()` (*typepy.Nan method*), 22
- `validate()` (*typepy.NoneType method*), 23
- `validate()` (*typepy.NullString method*), 24
- `validate()` (*typepy.RealNumber method*), 25
- `validate()` (*typepy.String method*), 26