pyproprop Documentation

Sam Brockie

Sep 14, 2021

CONTENTS:

Pyproprop (from Python-processed-properties) is a Python package to help you write classes that contain lots of repetitive properties which implement idioms such as type checking, bounds checking, type casting, method applications etc. with ease and in a DRY manner.

New users of *Pyproprop* should check out the *User Manual*.

CHAPTER

ONE

USER MANUAL

1.1 Overview

Do you often find yourself writing classes with properties such as:

```
from some_other_module import DefaultObject, some_type

class ExampleClass:

    def __init__(self,
                 type_checked_value,
                 bounded_numeric_value,
                 specific_length_sequence_value,
                 obj_with_method_applied_value,
                 ):
        self.type_check_attr = type_checked_value
        self.bounded_numeric_attr = bounded_numeric_value
        self.specific_length_sequence_attr = specific_length_sequence_value
        self.obj_with_method_applied_attr = obj_with_method_applied_value
        self.instantiate_default_if_none_attr = None

    @property
    def type_checked_attr(self):
        return self._type_checked_attr

    @type_checked_attr.setter
    def type_checked_attr(self, val):
        if not isinstance(val, some_type):
            msg = "`type_checked_attr` must be of `some_type`"
            raise TypeError(msg)
        self._type_checked_attr = val

    @property
    def bounded_numeric_attr(self):
        return self._bounded_numeric_attr

    @bounded_numeric_attr.setter
    def bounded_numeric_attr(self, val):
        val = float(val)
        lower_bound = -1.0
```

(continues on next page)

(continued from previous page)

```
        upper_bound = 2.5
        if val < lower_bound:
            msg = f"`bounded_numeric_attr` must be greater than {lower_bound}"
            raise ValueError(msg)
        if val >= upper_bound:
            msg = (f"`bounded_numeric_attr` must be less than or equal to "
                   f"{upper_bound}.")
            raise ValueError(msg)
        self._type_checked_attr = val

    @property
    def specific_length_sequence_attr(self):
        return self._specific_length_sequence_attr

    @specific_length_sequence_attr.setter
    def specific_length_sequence_attr(self, val):
        if len(val) != 2:
            msg = "`specific_length_sequence` must be an iterable of length 2."
            raise ValueError(msg)
        self._specific_length_sequence_attr = val

    @property
    def obj_with_method_applied_value(self):
        return self._obj_with_method_applied_value

    @obj_with_method_applied_value.setter
    def obj_with_method_applied_value(self, val):
        val = str(val)
        self._obj_with_method_applied_value = val.title()

    @property
    def instantiate_default_if_none_attr(self):
        return self._instantiate_default_if_none_attr

    @instantiate_default_if_none_attr.setter
    def instantiate_default_if_none_attr(self, val):
        if val is None:
            val = DefaultObject()
        self._instantiate_default_if_none_attr = val
```

With *Pyproprop* all of this boilerplate can be removed and instead the exact same class can be rewritten as:

```
from pyproprop import processed_property
from some_other_module import DefaultObject, some_type

class ExampleClass:

    type_checked_attr = processed_property(
        "type_checked_attr",
        description="property with enforced type of `some_type`",
        type=some_type,
    )
```

(continues on next page)

(continued from previous page)

```
    bounded_numeric_attr = processed_property(
        "bounded_numeric_attr",
        description="numerical attribute with upper and lower bounds"
        type=float,
        cast=True,
        min=-1.0,
        max=2.5,
    )
    specific_length_sequence_attr = processed_property(
        "specific_length_sequence_attr",
        description="sequence of length exactly 2",
        len=2,
    )
    obj_with_method_applied_attr = processed_property(
        "obj_with_method_applied_attr",
        description="sting formatted to use title case"
        type=str,
        cast=True,
        method="title",
    )
    instantiate_default_if_none_attr = processed_property(
        "instantiate_default_if_none_attr",
        default=DefaultObject,
    )

    def __init__(self,
                 type_checked_value,
                 bounded_numeric_value,
                 specific_length_sequence_value,
                 obj_with_method_applied_value,
                 ):
        self.type_check_attr = type_checked_value
        self.bounded_numeric_attr = bounded_numeric_value
        self.specific_length_sequence_attr = specific_length_sequence_value
        self.obj_with_method_applied_attr = obj_with_method_applied_value
        self.instantiate_default_if_none_attr = None
```

1.2 Installation

The easiest way to install *Pyproprop* is using the Anaconda Python distribution and its included *Conda* package management system. To install *Pyproprop* and its required dependencies, enter the following command at a command prompt:

```
conda install pyproprop
```

To install using *pip*, enter the following command at a command prompt:

```
pip install pyproprop
```

1.3 Recipes

CHAPTER

TWO

DEVELOPMENT MANUAL

2.1 Contributing

Source code for *Pyproprop* is located at https://github.com/brocksam/pyproprop. If you wish to submit a bug report, enhancement or feature request please create an issue using the GitHub issue tracker https://github.com/brocksam/pyproprop/issues.

2.2 Style Guide

2.3 Repository Map

2.4 Project To Do

CHAPTER

THREE

API DOCUMENTATION

CHAPTER

FOUR

PYPROPROP

4.1 pyproprop package

4.1.1 Submodules

4.1.2 pyproprop.format_str_case module

4.1.3 pyproprop.named_iterable module

4.1.4 pyproprop.options module

4.1.5 pyproprop.processed_property module

4.1.6 pyproprop.utils module

4.1.7 Module contents

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search