Property Definitions (bpy.props)

This module defines properties to extend Blender's internal data. The result of these functions is used to assign properties to classes registered with Blender and can't be used directly.

Note

All parameters to these functions must be passed as keywords.

Assigning to Existing Classes

Custom properties can be added to any subclass of an `ID`, `Bone` and `PoseBone`.

These properties can be animated, accessed by the user interface and python like Blender's existing properties.

Warning

Access to these properties might happen in threaded context, on a per-data-block level. This has to be carefully considered when using accessors or update callbacks.

Typically, these callbacks should not affect any other data that the one owned by their data-block. When accessing external non-Blender data, thread safety mechanisms should be considered.

```
import bpy

# Assign a custom property to an existing type.
bpy.types.Material.custom_float = bpy.props.FloatProperty(name="Test Property")

# Test the property is there.
bpy.data.materials[0].custom_float = 5.0
```

Operator Example

A common use of custom properties is for python based `Operator` classes. Test this code by running it in the text editor, or by clicking the button in the 3D View-port's Tools panel. The latter will show the properties in the Redo panel and allow you to change them.

```
import bpy


class OBJECT_OT_property_example(bpy.types.Operator):
    bl_idname = "object.property_example"
    bl_label = "Property Example"
    bl_options = {'REGISTER', 'UNDO'}

    my_float: bpy.props.FloatProperty(name="Some Floating Point")
    my_bool: bpy.props.BoolProperty(name="Toggle Option")
    my_string: bpy.props.StringProperty(name="String Value")

    def execute(self, context):
        self.report(
            {'INFO'}, "F: {:.2f}  B: {:s}  S: {!r}".format(
                self.my_float, self.my_bool, self.my_string,
            )
        )
        print('My float:', self.my_float)
```

```
        print('My bool:', self.my_bool)
        print('My string:', self.my_string)
        return {'FINISHED'}


class OBJECT_PT_property_example(bpy.types.Panel):
    bl_idname = "object_PT_property_example"
    bl_label = "Property Example"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = "Tool"

    def draw(self, context):
        # You can set the property values that should be used when the user
        # presses the button in the UI.
        props = self.layout.operator('object.property_example')
        props.my_bool = True
        props.my_string = "Shouldn't that be 47?"

        # You can set properties dynamically:
        if context.object:
            props.my_float = context.object.location.x
        else:
            props.my_float = 327


bpy.utils.register_class(OBJECT_OT_property_example)
bpy.utils.register_class(OBJECT_PT_property_example)

# Demo call. Be sure to also test in the 3D Viewport.
bpy.ops.object.property_example(
    my_float=47,
    my_bool=True,
    my_string="Shouldn't that be 327?",
)
```

PropertyGroup Example

PropertyGroups can be used for collecting custom settings into one value to avoid many individual settings mixed in together.

```
import bpy


class MaterialSettings(bpy.types.PropertyGroup):
    my_int: bpy.props.IntProperty()
    my_float: bpy.props.FloatProperty()
    my_string: bpy.props.StringProperty()


bpy.utils.register_class(MaterialSettings)

bpy.types.Material.my_settings = bpy.props.PointerProperty(type=MaterialSettings)

# test the new settings work
```

```
material = bpy.data.materials[0]

material.my_settings.my_int = 5
material.my_settings.my_float = 3.0
material.my_settings.my_string = "Foo"
```

Collection Example

Custom properties can be added to any subclass of an `ID`, `Bone` and `PoseBone`.

```
import bpy


# Assign a collection.
class SceneSettingItem(bpy.types.PropertyGroup):
    name: bpy.props.StringProperty(name="Test Property", default="Unknown")
    value: bpy.props.IntProperty(name="Test Property", default=22)


bpy.utils.register_class(SceneSettingItem)

bpy.types.Scene.my_settings = bpy.props.CollectionProperty(type=SceneSettingItem)

# Assume an armature object selected.
print("Adding 2 values!")

my_item = bpy.context.scene.my_settings.add()
my_item.name = "Spam"
my_item.value = 1000

my_item = bpy.context.scene.my_settings.add()
my_item.name = "Eggs"
my_item.value = 30

for my_item in bpy.context.scene.my_settings:
    print(my_item.name, my_item.value)
```

Update Example

It can be useful to perform an action when a property is changed and can be used to update other properties or synchronize with external data.

All properties define update functions except for CollectionProperty.

> Warning
>
> Remember that these callbacks may be executed in threaded context.

> Warning
>
> If the property belongs to an Operator, the update callback's first parameter will be an OperatorProperties instance, rather than an instance of the operator itself. This means you can't access other internal functions of the operator, only its other properties.

```
import bpy


def update_func(self, context):
```

```
    print("my test function", self)


bpy.types.Scene.testprop = bpy.props.FloatProperty(update=update_func)

bpy.context.scene.testprop = 11.0

# >>> my test function <bpy_struct, Scene("Scene")>
```

Getter/Setter Example

Getter/setter functions can be used for boolean, int, float, string and enum properties. If these callbacks are defined the property will not be stored in the ID properties automatically. Instead, the `get` and `set` functions will be called when the property is respectively read or written from the API.

> **Warning**
>
> Remember that these callbacks may be executed in threaded context.

```
import bpy


# Simple property reading/writing from ID properties.
# This is what the RNA would do internally.
def get_float(self):
    return self["testprop"]


def set_float(self, value):
    self["testprop"] = value


bpy.types.Scene.test_float = bpy.props.FloatProperty(get=get_float, set=set_float)


# Read-only string property, returns the current date
def get_date(self):
    import datetime
    return str(datetime.datetime.now())


bpy.types.Scene.test_date = bpy.props.StringProperty(get=get_date)


# Boolean array. Set function stores a single boolean value, returned as the second compon
# Array getters must return a list or tuple
# Array size must match the property vector size exactly
def get_array(self):
    return (True, self["somebool"])


def set_array(self, values):
    self["somebool"] = values[0] and values[1]


bpy.types.Scene.test_array = bpy.props.BoolVectorProperty(size=2, get=get_array, set=set_a
```

```
# Enum property.
# Note: the getter/setter callback must use integer identifiers!
test_items = [
    ("RED", "Red", "", 1),
    ("GREEN", "Green", "", 2),
    ("BLUE", "Blue", "", 3),
    ("YELLOW", "Yellow", "", 4),
]


def get_enum(self):
    import random
    return random.randint(1, 4)


def set_enum(self, value):
    print("setting value", value)


bpy.types.Scene.test_enum = bpy.props.EnumProperty(items=test_items, get=get_enum, set=set


# Testing the properties:
scene = bpy.context.scene

scene.test_float = 12.34
print('test_float:', scene.test_float)

scene.test_array = (True, False)
print('test_array:', tuple(scene.test_array))

# scene.test_date = "blah"   # this would fail, property is read-only
print('test_date:', scene.test_date)

scene.test_enum = 'BLUE'
print('test_enum:', scene.test_enum)

# The above outputs:
# test_float: 12.34000015258789
# test_array: (True, False)
# test_date: 2018-03-14 11:36:53.158653
# setting value 3
# test_enum: GREEN
```

bpy.props.**BoolProperty(*, name='', description='', translation_context='*', default=False, options={'ANIMATABLE'}, override=set(), tags=set(), subtype='NONE', update=None, get=None, set=None)**

Returns a new boolean property definition.

PARAMETERS:

- **name** (*str*) – Name used in the user interface.
- **description** (*str*) – Text used for the tooltip and api documentation.
- **translation_context** (*str*) – Text used as context to disambiguate translations.

- **options** (*set[str]*) – Enumerator in Property Flag Items.
- **override** (*set[str]*) – Enumerator in Property Override Flag Items.
- **tags** (*set[str]*) – Enumerator of tags that are defined by parent class.
- **subtype** (*str*) – Enumerator in Property Subtype Number Items.
- **update** (Callable[[`bpy.types.bpy_struct`, `bpy.types.Context`], None]) – Function to be called when this value is modified, This function must take 2 values (self, context) and return None. *Warning* there are no safety checks to avoid infinite recursion.
- **get** (Callable[[`bpy.types.bpy_struct`], bool]) – Function to be called when this value is 'read', This function must take 1 value (se and return the value of the property.
- **set** (Callable[[`bpy.types.bpy_struct`, bool], None]) – Function to be called when this value is 'written', This function must take 2 values (self, value) and return None.

bpy.props.**BoolVectorProperty(*, name='', description='', translation_context='*', default=(False, False, False), options={'ANIMATABLE'}, override=set(), tags=set(), subtype='NONE', size=3, update=None, get=None, set=None)**

Returns a new vector boolean property definition.

PARAMETERS:

- **name** (*str*) – Name used in the user interface.
- **description** (*str*) – Text used for the tooltip and api documentation.
- **translation_context** (*str*) – Text used as context to disambiguate translations.
- **default** (*Sequence[bool]*) – sequence of booleans the length of *size*.
- **options** (*set[str]*) – Enumerator in Property Flag Items.
- **override** (*set[str]*) – Enumerator in Property Override Flag Items.
- **tags** (*set[str]*) – Enumerator of tags that are defined by parent class.
- **subtype** (*str*) – Enumerator in Property Subtype Number Array Items.
- **size** (*int* | *Sequence[int]*) – Vector dimensions in [1, 32]. An int sequence can be used to define multi-dimension arrays.
- **update** (Callable[[`bpy.types.bpy_struct`, `bpy.types.Context`], None]) – Function to be called when this value is modified, This function must take 2 values (self, context) and return None. *Warning* there are no safety checks to avoid infinite recursion.
- **get** (Callable[[`bpy.types.bpy_struct`], Sequence[bool]]) – Function to be called when this value is 'read', This function must take value (self) and return the value of the property.
- **set** (Callable[[`bpy.types.bpy_struct`, tuple[bool, …]], None]) – Function to be called when this value is 'written', This function must take 2 values (self, value) and return None.

bpy.props.**CollectionProperty(type=None, *, name='', description='', translation_context='*', options={'ANIMATABLE'}, override=set(), tags=set())**

Returns a new collection property definition.

PARAMETERS:

- **type** (type[`bpy.types.PropertyGroup`]) – A subclass of a property group.
- **name** (*str*) – Name used in the user interface.
- **description** (*str*) – Text used for the tooltip and api documentation.
- **translation_context** (*str*) – Text used as context to disambiguate translations.
- **options** (*set[str]*) – Enumerator in Property Flag Items.
- **override** (*set[str]*) – Enumerator in Property Override Flag Collection Items.
- **tags** (*set[str]*) – Enumerator of tags that are defined by parent class.

bpy.props.**EnumProperty(items, *, name='', description='', translation_context='*', default=None, options={'ANIMATABLE'}, override=set(), tags=set(), update=None, get=None, set=None)**

Returns a new enumerator property definition.

PARAMETERS:

- **items** (Sequence[tuple[str, str, str] | tuple[str, str, str, int] | tuple[str, str, str, int, int] | None] | Callable[[`bpy.types.bpy_struct`, `bpy.types.Context` | None], Sequence[tuple[str, str, str] | tuple[str, str, str, int] | tuple[str, str, str, int, int] | None]]) – sequence of enum items formatted: `[(identifier, name, description, icon, number), ...]`.

The first three elements of the tuples are mandatory.

IDENTIFIER:

The identifier is used for Python access. An empty identifier means that the item is a separator

NAME:

Name for the interface.

DESCRIPTION:

Used for documentation and tooltips.

ICON:

An icon string identifier or integer icon value (e.g. returned by `bpy.types.UILayout.icon`)

NUMBER:

Unique value used as the identifier for this item (stored in file data). Use when the identifier may need to change. If the *ENUM_FLAG* option is used, the values are bit-masks and should be powers of two.

When an item only contains 4 items they define `(identifier, name, description, number)`.

Separators may be added using either None (nameless separator), or a regular item tuple with an empty identifier string, in which case the nam if non-empty, will be displayed in the UI above the separator line. For dynamic values a callback can be passed which returns a list in the sam format as the static list. This function must take 2 arguments `(self, context)`, **context may be None**.

Warning

There is a known bug with using a callback, Python must keep a reference to the strings returned by the callback or Blender will misbehave or even crash.

- **name** (*str*) – Name used in the user interface.
- **description** (*str*) – Text used for the tooltip and api documentation.
- **translation_context** (*str*) – Text used as context to disambiguate translations.
- **default** (*str | int | set[str]*) – The default value for this enum, a string from the identifiers used in *items*, or integer matching an item number. I the *ENUM_FLAG* option is used this must be a set of such string identifiers instead. WARNING: Strings cannot be specified for dynamic enums (i.e. if a callback function is given as *items* parameter).
- **options** (*set[str]*) – Enumerator in Property Flag Enum Items.
- **override** (*set[str]*) – Enumerator in Property Override Flag Items.
- **tags** (*set[str]*) – Enumerator of tags that are defined by parent class.
- **update** (Callable[[`bpy.types.bpy_struct`, `bpy.types.Context`], None]) – Function to be called when this value is modified, This function must take 2 values (self, context) and return None. *Warning* there are no safety checks to avoid infinite recursion.
- **get** (Callable[[`bpy.types.bpy_struct`], int]) – Function to be called when this value is 'read', This function must take 1 value (self) and return the value of the property.
- **set** (Callable[[`bpy.types.bpy_struct`, int], None]) – Function to be called when this value is 'written', This function must take 2 values (self, value) and return None.

bpy.props.**FloatProperty(*, name='', description='', translation_context='*', default=0.0, min=-3.402823e+38, max=3.402823e+38, soft_min=-3.402823e+38, soft_max=3.402823e+38, step=3, precision=2, options={'ANIMATABLE'}, override=set(), tags=set(), subtype='NONE', unit='NONE', update=None, get=None, set=None)**

Returns a new float (single precision) property definition.

PARAMETERS:

- **name** (*str*) – Name used in the user interface.
- **description** (*str*) – Text used for the tooltip and api documentation.
- **translation_context** (*str*) – Text used as context to disambiguate translations.
- **min** (*float*) – Hard minimum, trying to assign a value below will silently assign this minimum instead.
- **max** (*float*) – Hard maximum, trying to assign a value above will silently assign this maximum instead.
- **soft_min** (*float*) – Soft minimum (>= *min*), user won't be able to drag the widget below this value in the UI.

- **soft_max** (*float*) – Soft maximum (<= *max*), user won't be able to drag the widget above this value in the UI.
- **step** (*int*) – Step of increment/decrement in UI, in [1, 100], defaults to 3 (WARNING: actual value is /100).
- **precision** (*int*) – Maximum number of decimal digits to display, in [0, 6]. Fraction is automatically hidden for exact integer values of fields wit unit 'NONE' or 'TIME' (frame count) and step divisible by 100.
- **options** (*set[str]*) – Enumerator in Property Flag Items.
- **override** (*set[str]*) – Enumerator in Property Override Flag Items.
- **tags** (*set[str]*) – Enumerator of tags that are defined by parent class.
- **subtype** (*str*) – Enumerator in Property Subtype Number Items.
- **unit** (*str*) – Enumerator in Property Unit Items.
- **update** (Callable[[`bpy.types.bpy_struct`, `bpy.types.Context`], None]) – Function to be called when this value is modified, This function must take 2 values (self, context) and return None. *Warning* there are no safety checks to avoid infinite recursion.
- **get** (Callable[[`bpy.types.bpy_struct`], float]) – Function to be called when this value is 'read', This function must take 1 value (se and return the value of the property.
- **set** (Callable[[`bpy.types.bpy_struct`, float], None]) – Function to be called when this value is 'written', This function must take 2 values (self, value) and return None.

bpy.props.**FloatVectorProperty(*, name='', description='', translation_context='*', default=(0.0, 0.0, 0.0), min=sys.float_info.min, max=sys.float_info.max, soft_min=sys.float_info.min, soft_max=sys.float_info.max, step=3, precision=2, options={'ANIMATABLE'}, override=set(), tags=set(), subtype='NONE', unit='NONE', size=3, update=None, get=None, set=None)**

Returns a new vector float property definition.

PARAMETERS:

- **name** (*str*) – Name used in the user interface.
- **description** (*str*) – Text used for the tooltip and api documentation.
- **translation_context** (*str*) – Text used as context to disambiguate translations.
- **default** (*Sequence[float]*) – Sequence of floats the length of *size*.
- **min** (*float*) – Hard minimum, trying to assign a value below will silently assign this minimum instead.
- **max** (*float*) – Hard maximum, trying to assign a value above will silently assign this maximum instead.
- **soft_min** (*float*) – Soft minimum (>= *min*), user won't be able to drag the widget below this value in the UI.
- **soft_max** (*float*) – Soft maximum (<= *max*), user won't be able to drag the widget above this value in the UI.
- **options** (*set[str]*) – Enumerator in Property Flag Items.
- **override** (*set[str]*) – Enumerator in Property Override Flag Items.
- **tags** (*set[str]*) – Enumerator of tags that are defined by parent class.
- **step** (*int*) – Step of increment/decrement in UI, in [1, 100], defaults to 3 (WARNING: actual value is /100).
- **precision** (*int*) – Maximum number of decimal digits to display, in [0, 6]. Fraction is automatically hidden for exact integer values of fields wit unit 'NONE' or 'TIME' (frame count) and step divisible by 100.
- **subtype** (*str*) – Enumerator in Property Subtype Number Array Items.
- **unit** (*str*) – Enumerator in Property Unit Items.
- **size** (*int* | *Sequence[int]*) – Vector dimensions in [1, 32]. An int sequence can be used to define multi-dimension arrays.
- **update** (Callable[[`bpy.types.bpy_struct`, `bpy.types.Context`], None]) – Function to be called when this value is modified, This function must take 2 values (self, context) and return None. *Warning* there are no safety checks to avoid infinite recursion.
- **get** (Callable[[`bpy.types.bpy_struct`], Sequence[float]]) – Function to be called when this value is 'read', This function must take value (self) and return the value of the property.
- **set** (Callable[[`bpy.types.bpy_struct`, tuple[float, …]], None]) – Function to be called when this value is 'written', This function must take 2 values (self, value) and return None.

bpy.props.**IntProperty(*, name='', description='', translation_context='*', default=0, min=-2**31, max=2**31 - 1, soft_min=-2**31, soft_max=2**31 - 1, step=1, options={'ANIMATABLE'}, override=set(), tags=set(), subtype='NONE', update=None, get=None, set=None)**

Returns a new int property definition.

PARAMETERS:

- **name** (*str*) – Name used in the user interface.
- **description** (*str*) – Text used for the tooltip and api documentation.
- **translation_context** (*str*) – Text used as context to disambiguate translations.
- **min** (*int*) – Hard minimum, trying to assign a value below will silently assign this minimum instead.
- **max** (*int*) – Hard maximum, trying to assign a value above will silently assign this maximum instead.
- **soft_min** (*int*) – Soft minimum (>= *min*), user won't be able to drag the widget below this value in the UI.
- **soft_max** (*int*) – Soft maximum (<= *max*), user won't be able to drag the widget above this value in the UI.
- **step** (*int*) – Step of increment/decrement in UI, in [1, 100], defaults to 1 (WARNING: unused currently!).
- **options** (*set[str]*) – Enumerator in Property Flag Items.
- **override** (*set[str]*) – Enumerator in Property Override Flag Items.
- **tags** (*set[str]*) – Enumerator of tags that are defined by parent class.
- **subtype** (*str*) – Enumerator in Property Subtype Number Items.
- **update** (Callable[[`bpy.types.bpy_struct`, `bpy.types.Context`], None]) – Function to be called when this value is modified, This function must take 2 values (self, context) and return None. *Warning* there are no safety checks to avoid infinite recursion.
- **get** (Callable[[`bpy.types.bpy_struct`], int]) – Function to be called when this value is 'read', This function must take 1 value (self) and return the value of the property.
- **set** (Callable[[`bpy.types.bpy_struct`, int], None]) – Function to be called when this value is 'written', This function must take 2 values (self, value) and return None.

bpy.props.**IntVectorProperty(*, name='', description='', translation_context='*', default=(0, 0, 0), min=-2**31, max=2**31 - 1, soft_min=-2**31, soft_max=2**31 - 1, step=1, options={'ANIMATABLE'}, override=set(), tags=set(), subtype='NONE', size=3, update=None, get=None, set=None)**

Returns a new vector int property definition.

PARAMETERS:

- **name** (*str*) – Name used in the user interface.
- **description** (*str*) – Text used for the tooltip and api documentation.
- **translation_context** (*str*) – Text used as context to disambiguate translations.
- **default** (*Sequence[int]*) – sequence of ints the length of *size*.
- **min** (*int*) – Hard minimum, trying to assign a value below will silently assign this minimum instead.
- **max** (*int*) – Hard maximum, trying to assign a value above will silently assign this maximum instead.
- **soft_min** (*int*) – Soft minimum (>= *min*), user won't be able to drag the widget below this value in the UI.
- **soft_max** (*int*) – Soft maximum (<= *max*), user won't be able to drag the widget above this value in the UI.
- **step** (*int*) – Step of increment/decrement in UI, in [1, 100], defaults to 1 (WARNING: unused currently!).
- **options** (*set[str]*) – Enumerator in Property Flag Items.
- **override** (*set[str]*) – Enumerator in Property Override Flag Items.
- **tags** (*set[str]*) – Enumerator of tags that are defined by parent class.
- **subtype** (*str*) – Enumerator in Property Subtype Number Array Items.
- **size** (*int* | *Sequence[int]*) – Vector dimensions in [1, 32]. An int sequence can be used to define multi-dimension arrays.
- **update** (Callable[[`bpy.types.bpy_struct`, `bpy.types.Context`], None]) – Function to be called when this value is modified, This function must take 2 values (self, context) and return None. *Warning* there are no safety checks to avoid infinite recursion.
- **get** (Callable[[`bpy.types.bpy_struct`], Sequence[int]]) – Function to be called when this value is 'read', This function must take 1 value (self) and return the value of the property.
- **set** (Callable[[`bpy.types.bpy_struct`, tuple[int, ...]], None]) – Function to be called when this value is 'written', This function must take 2 values (self, value) and return None.

bpy.props.**PointerProperty(type=None, *, name='', description='', translation_context='*', options={'ANIMATABLE'}, override=set(), tags=set(), poll=None, update=None)**

Returns a new pointer property definition.

PARAMETERS:

PARAMETERS:

- **type** (type[`bpy.types.PropertyGroup` | `bpy.types.ID`]) – A subclass of a property group or ID types.
- **name** (*str*) – Name used in the user interface.
- **description** (*str*) – Text used for the tooltip and api documentation.
- **translation_context** (*str*) – Text used as context to disambiguate translations.
- **options** (*set[str]*) – Enumerator in Property Flag Items.
- **override** (*set[str]*) – Enumerator in Property Override Flag Items.
- **tags** (*set[str]*) – Enumerator of tags that are defined by parent class.
- **poll** (Callable[[`bpy.types.bpy_struct` , `bpy.types.ID`], bool]) –
  Function that determines whether an item is valid for this property. The function must take 2 values (self, object) and return a boolean.

  Note

  The return value will be checked only when assigning an item from the UI, but it is still possible to assign an "invalid" item to the property directly.

- **update** (Callable[[`bpy.types.bpy_struct` , `bpy.types.Context`], None]) – Function to be called when this value is modified, This function must take 2 values (self, context) and return None. *Warning* there are no safety checks to avoid infinite recursion.

Note

Pointer properties do not support storing references to embedded IDs (e.g. *bpy.types.Scene.collection*, *bpy.types.Material.node_tree*). These should exclusively be referenced and accessed through their owner ID (e.g. the scene or material).

bpy.props.**RemoveProperty(cls, attr)**

Removes a dynamically defined property.

PARAMETERS:

- **cls** (*type*) – The class containing the property (must be a positional argument).
- **attr** (*str*) – Property name (must be passed as a keyword).

Note

Typically this function doesn't need to be accessed directly. Instead use `del cls.attr`

bpy.props.**StringProperty(*, name='', description='', translation_context='*', default='', maxlen=0, options={'ANIMATABLE'}, override=set(), tags=set(), subtype='NONE', update=None, get=None, set=None, search=None, search_options={'SUGGESTION'})**

Returns a new string property definition.

PARAMETERS:

- **name** (*str*) – Name used in the user interface.
- **description** (*str*) – Text used for the tooltip and api documentation.
- **translation_context** (*str*) – Text used as context to disambiguate translations.
- **default** (*str*) – initializer string.
- **maxlen** (*int*) – maximum length of the string.
- **options** (*set[str]*) – Enumerator in Property Flag Items.
- **override** (*set[str]*) – Enumerator in Property Override Flag Items.
- **tags** (*set[str]*) – Enumerator of tags that are defined by parent class.
- **subtype** (*str*) – Enumerator in Property Subtype String Items.
- **update** (Callable[[`bpy.types.bpy_struct` , `bpy.types.Context`], None]) – Function to be called when this value is modified, This function must take 2 values (self, context) and return None. *Warning* there are no safety checks to avoid infinite recursion.
- **get** (Callable[[`bpy.types.bpy_struct`], str]) – Function to be called when this value is 'read', This function must take 1 value (self) and return the value of the property.
- **set** (Callable[[`bpy.types.bpy_struct` , str], None]) – Function to be called when this value is 'written', This function must take 2

values (self, value) and return None.

- **search** (Callable[[`bpy.types.bpy_struct` , `bpy.types.Context` , str], Iterable[str | tuple[str, str]]]) –

  Function to be called to show candidates for this string (shown in the UI). This function must take 3 values (self, context, edit_text) and return sequence, iterator or generator where each item must be:

  - A single string (representing a candidate to display).
  - A tuple-pair of strings, where the first is a candidate and the second is additional information about the candidate.

- **search_options** (*set[str]*) –

  Set of strings in:

  - 'SORT' sorts the resulting items.
  - 'SUGGESTION' lets the user enter values not found in search candidates. **WARNING** disabling this flag causes the search callback to r
    on redraw, so only disable this flag if it's not likely to cause performance issues.