

---

# **Anchorpoint**

*Release 0.7.0*

**Matt Carey**

**Oct 18, 2021**



## CONTENTS:

<b>1</b>	<b>Selecting Text with Anchorpoint</b>	<b>3</b>
1.1	Converting Between Selector Types . . . . .	4
1.2	Combining and Grouping Selectors . . . . .	4
1.3	Comparing Selectors and Sets . . . . .	5
1.4	Serializing Selectors . . . . .	5
<b>2</b>	<b>Development Updates</b>	<b>7</b>
2.1	Changelog . . . . .	7
2.1.1	0.7.0 (2021-10-09) . . . . .	7
2.1.2	0.6.1 (2021-09-23) . . . . .	7
2.1.3	0.6.0 (2021-09-19) . . . . .	7
2.1.4	0.5.3 (2021-08-11) . . . . .	8
2.1.5	0.5.2 (2021-08-02) . . . . .	8
2.1.6	0.5.1 (2021-05-15) . . . . .	8
2.1.7	0.5.0 (2021-05-07) . . . . .	8
2.1.8	0.4.4 (2021-01-25) . . . . .	8
2.1.9	0.4.3 (2020-12-11) . . . . .	8
2.1.10	0.4.2 (2020-08-30) . . . . .	9
2.1.11	0.4.1 (2020-08-29) . . . . .	9
2.1.12	0.4.0 (2020-08-08) . . . . .	9
2.1.13	0.3.3 (2020-07-28) . . . . .	9
2.1.14	0.3.2 (2020-07-22) . . . . .	9
2.1.15	0.3.1 (2020-07-19) . . . . .	9
2.1.16	0.3.0 (2020-07-18) . . . . .	9
2.1.17	0.2.1 (2020-05-21) . . . . .	10
2.1.18	0.2.0 (2020-05-21) . . . . .	10
2.1.19	0.1.1 (2019-12-01) . . . . .	10
2.1.20	0.1.0 (2019-11-30) . . . . .	10
2.2	GitHub . . . . .	10
2.3	Twitter . . . . .	10
<b>3</b>	<b>API Reference</b>	<b>11</b>
3.1	Text Selectors . . . . .	11
3.2	Text Sequences . . . . .	19
<b>4</b>	<b>Indices and tables</b>	<b>21</b>
	<b>Python Module Index</b>	<b>23</b>
	<b>Index</b>	<b>25</b>



Release v. 0.7.0.

Anchorpoint is a Python library that supplies text substring selectors for anchoring annotations. It includes helper methods for switching between positional and contextual selectors, and it's used for referencing judicial opinions and statutes by [AuthoritySpoke](#).

Anchorpoint is licensed under the [Atmosphere Software License](#).



## SELECTING TEXT WITH ANCHORPOINT

Anchorpoint is a tool for labeling referenced passages within text documents, in a format that allows the “anchors” to the referenced passages to be stored and transmitted separately from the documents themselves. Anchorpoint has two basic ways of selecting text: as text positions, or as text quotes. Here’s a demonstration of creating a text string in Python and then using both kinds of text selectors.

```
>>> from anchorpoint import TextPositionSelector, TextQuoteSelector
>>> legal_text = (
...     "Copyright protection subsists, in accordance with this title, "
...     "in original works of authorship fixed in any tangible medium of expression, "
...     "now known or later developed, from which they can be perceived, reproduced, "
...     "or otherwise communicated, either directly or with the aid of a machine or
↳device. "
...     "Works of authorship include the following categories: "
...     "literary works; musical works, including any accompanying words; "
...     "dramatic works, including any accompanying music; "
...     "pantomimes and choreographic works; "
...     "pictorial, graphic, and sculptural works; "
...     "motion pictures and other audiovisual works; "
...     "sound recordings; and architectural works.")
>>> positions = TextPositionSelector(start=65, end=93)
>>> positions.select_text(legal_text)
'original works of authorship'
>>> quote = TextQuoteSelector(exact="in accordance with this title")
>>> quote.select_text(legal_text)
'in accordance with this title'
```

A *TextPositionSelector* works by identifying the positions of the start and end characters within the text string object, while a *TextQuoteSelector* describes the part of the text that is being selected.

Sometimes a selected passage is too long to include in full in the *TextQuoteSelector*. In that case, you can identify the selection by specifying its prefix and suffix. That is, the text immediately before and immediately after the text you want to select.

```
>>> quote = TextQuoteSelector(prefix="otherwise communicated, ", suffix=" Works of
↳authorship")
>>> quote.select_text(legal_text)
'either directly or with the aid of a machine or device.'
```

If you specify just a suffix, then the start of your text selection is the beginning of the text string. If you specify just a prefix, then your text selection continues to the end of the text string.

```
>>> quote_from_start = TextQuoteSelector(suffix="in accordance with this title")
>>> quote_from_start.select_text(legal_text)
'Copyright protection subsists,'
>>> quote_from_end = TextQuoteSelector(prefix="sound recordings; and")
>>> quote_from_end.select_text(legal_text)
'architectural works.'
```

If you want to use a *TextQuoteSelector* to select a particular instance of a phrase that appears more than once in the text, then you can add a prefix or suffix in addition to the exact phrase to eliminate the ambiguity. For example, this selector applies to the second instance of the word “authorship” in the text, not the first instance.

```
>>> authorship_selector = TextQuoteSelector(exact="authorship", suffix="include")
>>> authorship_selector.select_text(legal_text)
'authorship'
```

## 1.1 Converting Between Selector Types

You can use the *as\_position()* and *as\_quote()* methods to convert between the two types of selector.

```
>>> authorship_selector.as_position(legal_text)
TextPositionSelector(start=306, end=316)
>>> positions.as_quote(legal_text)
TextQuoteSelector(exact='original works of authorship', prefix='', suffix='')
```

## 1.2 Combining and Grouping Selectors

Position selectors can be combined into a single selector that covers both spans of text.

```
>>> left = TextPositionSelector(start=5, end=22)
>>> right = TextPositionSelector(start=12, end=27)
>>> left + right
TextPositionSelector(start=5, end=27)
```

If two position selectors don’t overlap, then adding them returns a different class called a *TextPositionSet*.

```
>>> from anchorpoint import TextPositionSet
>>> left = TextPositionSelector(start=65, end=79)
>>> right = TextPositionSelector(start=100, end=136)
>>> selector_set = left + right
>>> selector_set
TextPositionSet(positions=[TextPositionSelector(start=65, end=79),
↳TextPositionSelector(start=100, end=136)], quotes=[])
```

The *TextPositionSet* can be used to select nonconsecutive passages of text.

```
>>> selector_set.select_text(legal_text)
'...original works...in any tangible medium of expression...'
```

If needed, you can use a *TextPositionSet* to select text with a combination of both positions and quotes.



```
>>> text = "red orange yellow green blue indigo violet"
>>> position = TextPositionSelector(start=4, end=17)
>>> quote = TextQuoteSelector(exact="blue indigo")
>>> group = TextPositionSet(positions=[position], quotes=[quote])
>>> group.select_text(text)
'...orange yellow...blue indigo...'
```

You can also add or subtract an integer to move the text selection left or right, but only the position selectors will be moved, not the quote selectors.

```
>>> earlier_selectors = group - 7
>>> earlier_selectors.select_text(text)
'red orange...blue indigo...'
```

Union and intersection operators also work.

```
>>> left = TextPositionSelector(start=2, end=10)
>>> right = TextPositionSelector(start=5, end=20)
>>> left & right
TextPositionSelector(start=5, end=10)
```

## 1.3 Comparing Selectors and Sets

The greater than and less than operators can be used to check whether one selector or set covers the entire range of another. This is used to check whether one selector only contains text that's already within another selector.

```
>>> smaller = TextPositionSelector(start=4, end=8)
>>> overlapping = TextPositionSelector(start=6, end=50)
>>> overlapping > smaller
False
>>> superset = TextPositionSelector(start=0, end=10)
>>> superset > smaller
True
```

TextPositionSets also have a `__gt__()` method that works in the same way.

```
>>> selector_set > TextPositionSelector(start=100, end=110)
True
```

## 1.4 Serializing Selectors

Anchorpoint uses `Pydantic` to serialize selectors either to Python dictionaries or to JSON strings suitable for sending over the internet with APIs.

```
>>> authorship_selector.json()
'{"exact": "authorship", "prefix": "", "suffix": "include"}'
>>> selector_set.dict()
{'positions': [{'start': 65, 'end': 79}, {'start': 100, 'end': 136}], 'quotes': []}
```

Pydantic's data loading methods mean that you can also create the data for an Anchorpoint selector using nested dictionaries, and then load it with the class's constructor method.

```
>>> data = {'positions': [{'start': 65, 'end': 79}, {'start': 100, 'end': 136}]}
>>> TextPositionSet(**data)
TextPositionSet(positions=[TextPositionSelector(start=65, end=79),
↳TextPositionSelector(start=100, end=136)], quotes=[])
```

You can also [get a valid OpenAPI schema](#), for using Anchorpoint selectors in an API that you design.

```
>>> TextPositionSelector.schema_json()
'{"title": "TextPositionSelector", "description": "Describes a textual segment by start_
↳and end positions.\\n\\nBased on the Web Annotation Data Model `Text Position Selector\\
↳\\n<https://www.w3.org/TR/annotation-model/#text-position-selector>`_ standard\\n\\
↳n:param start:\\n    The starting position of the segment of text.\\n    The first_
↳character in the full text is character position 0,\\n    and the character is_
↳included within the segment.\\n\\n:param end:\\n    The end position of the segment of_
↳text.\\n    The character is not included within the segment.", "type": "object",
↳"properties": {"start": {"title": "Start", "default": 0, "type": "integer"}, "end": {
↳"title": "End", "type": "integer"}}}'
```

## DEVELOPMENT UPDATES

### 2.1 Changelog

#### 2.1.1 0.7.0 (2021-10-09)

- passing `TextPositionSet` to `from_selection_sequence` doesn't cause error
- fix bug: Range with end "Inf" caused string slicing error
- add `TextPositionSet.from_quotes`
- remove `TextSelector` class
- `start_less_than_end` is no longer a `root_validator`

#### 2.1.2 0.6.1 (2021-09-23)

- `python-ranges` by `Superbird11` is imported instead of vendored

#### 2.1.3 0.6.0 (2021-09-19)

- `TextPositionSelector` no longer inherits from `Range`
- `TextPositionSet` no longer inherits from `RangeSet`
- `TextPositionSelector` no longer has `real_start` and `real_end` that can differ from `start` and `end`
- `Selectors` and `TextPositionSets` are Pydantic models
- `TextSelector` is Pydantic model for either `Quote` or `Position Selector`
- remove `Marshmallow` schemas
- update type annotations for `TextPositionSelector.from_range`
- add `TestQuoteSelector.as_unique_position` method
- `TextPositionSet` can include `TextQuoteSelectors`
- add `convert_quotes_to_positions` method to `TextPositionSet`
- replace `TextPositionSet.selectors` field with `positions` and `quotes`
- change `as_quote_selector` method to `as_quote`
- `TextPositionSet.add_margin` includes quotes
- fix bug: subtracting int from selector set caused quotes to be lost

- add `__ge__` and `__gt__` methods for `TextPositionSelector`
- add Selecting Text with Anchorpoint guide

### 2.1.4 0.5.3 (2021-08-11)

- change readme to `.rst`
- use `setup.py` instead of `setup.cfg`

### 2.1.5 0.5.2 (2021-08-02)

- `TextPositionSet` can be made from list of tuples
- long passage in exception is truncated

### 2.1.6 0.5.1 (2021-05-15)

- improper shorthand for selector raises `TextSelectionError`

### 2.1.7 0.5.0 (2021-05-07)

- add `TextPositionSelector.from_text` constructor
- `Range` constructor interprets `None` as 0
- fix bug: union with `TextPositionSet` should return `TextPositionSet`
- add `PositionSelectorSchema`, for when a selector can't be a `TextQuoteSelector`

### 2.1.8 0.4.4 (2021-01-25)

- provide “missing” instead of “optional” argument for marshmallow schema
- add `TextPositionSetFactory.from_exact_strings`
- `SelectorSchema.expand_anchor_shorthand` takes only a string argument
- `TextPositionSetFactory.from_selection` will accept a Sequence of mixed types

### 2.1.9 0.4.3 (2020-12-11)

- `TextPositionSelector` serializer dumps `.real_start` and `.real_end`
- `TextPositionSelector` serializer omits “include\_start” and “include\_end”
- `TextPositionSelector` serializer orders fields so “start” comes before “end”
- disallow zero-length `TextPositionSelectors`

### 2.1.10 0.4.2 (2020-08-30)

- create `TextPositionSelector` `.real_start` and `.real_end`
- create `TextPositionSet.add_margin`

### 2.1.11 0.4.1 (2020-08-29)

- `TextPositionSetFactory` will accept list of strings
- subtracting more than start value is no longer `IndexError`, but more than end value is
- `TextSequence` quoting from empty string doesn't start with `None`

### 2.1.12 0.4.0 (2020-08-08)

- `TextPositionSet` can output a `TextSequence`
- create `TextSequence` addition method

### 2.1.13 0.3.3 (2020-07-28)

- fix bug: leading whitespace when selecting from prefix

### 2.1.14 0.3.2 (2020-07-22)

- fix bug where adding selectors converted them to parent class
- add `TextSelectionError` exception

### 2.1.15 0.3.1 (2020-07-19)

- add left and right margin parameters to `TextPositionSelector.as_quote_selector`
- `as_quotes` method for `TextSelectorSet`
- enable adding int to `TextSelectorSet`
- fix class name in repr for `TextSelectorSet`

### 2.1.16 0.3.0 (2020-07-18)

- add `TextQuoteSelector.from_text` shortcut
- add ability to subtract an integer from all values in a `TextPositionSet`
- include [marshmallow](<https://github.com/marshmallow-code/marshmallow>) schema for serializing

### **2.1.17 0.2.1 (2020-05-21)**

- add init file to utils directory

### **2.1.18 0.2.0 (2020-05-21)**

- Make TextPositionSelector subclass *Range* from [python-ranges](<https://github.com/Superbird11/ranges>).

### **2.1.19 0.1.1 (2019-12-01)**

- add init file to tests directory

### **2.1.20 0.1.0 (2019-11-30)**

- Create TextPositionSelector and TextQuoteSelector classes

## **2.2 GitHub**

You can find open issues and current changes to anchorpoint through its [GitHub repo](#).

## **2.3 Twitter**

On Twitter, you can follow [@authorityspoke](#) or [@mcareyaus](#) for project updates.

## 3.1 Text Selectors

Text substring selectors for anchoring annotations.

Based on parts of the W3C [Web Annotation Data Model](#).

**class** `anchorpoint.textselectors.TextPositionSelector`(*\*\*data*)

Describes a textual segment by start and end positions.

Based on the Web Annotation Data Model [Text Position Selector](#) standard

### Parameters

- **start** – The starting position of the segment of text. The first character in the full text is character position 0, and the character is included within the segment.
- **end** – The end position of the segment of text. The character is not included within the segment.

`__add__`(*value*)

Make a new selector covering the combined ranges of self and other.

### Parameters

- **other** – selector for another text interval
- **margin** – allowable distance between two selectors that can still be added together

**Return type** `Union[TextPositionSelector, TextPositionSet, None]`

**Returns** a selector reflecting the combined range if possible, otherwise None

`__and__`(*other*)

Make a new selector covering the intersection of the ranges of self and other.

**Parameters** **other** (`Union[TextPositionSelector, TextPositionSet, Range, RangeSet]`) – selector for another text interval

**Return type** `Optional[TextPositionSelector]`

**Returns** a selector reflecting the range of the intersection

`__ge__`(*other*)

Return self >= value.

**Return type** `bool`

`__gt__`(*other*)

Check if self is greater than other.

**Parameters** **other** (`Union[TextPositionSelector, TextPositionSet]`) – selector for another text interval

**Return type** `bool`

**Returns** whether self is greater than other

**\_\_hash\_\_** = `None`

**\_\_or\_\_**(*other*)

Make a new selector covering the combined ranges of self and other.

**Parameters** **other** (`Union[TextPositionSelector, TextPositionSet, Range, RangeSet]`) – selector for another text interval

**Return type** `Union[TextPositionSelector, TextPositionSet]`

**Returns** a selector reflecting the combined range

**as\_quote**(*text*, *left\_margin=0*, *right\_margin=0*)

Make a quote selector, creating prefix and suffix from specified lengths of text.

**Parameters**

- **text** (`str`) – the passage where an exact quotation needs to be located
- **left\_margin** (`int`) – number of characters to look backward to create `TextQuoteSelector.prefix`
- **right\_margin** (`int`) – number of characters to look forward to create `TextQuoteSelector.suffix`

**Return type** `TextQuoteSelector`

**combine**(*other*, *text*)

Make new selector combining ranges of self and other if it will fit in text.

**difference**(*other*)

Get selectors in self or other but not both.

Applies Range difference, method replacing `RangeSet` with `TextPositionSet` in return value.

**Return type** `Union[TextPositionSet, TextPositionSelector]`

**classmethod from\_range**(*range*)

Make `TextPositionSelector` with same extent as a `Range` object from python-ranges.

**Return type** `TextPositionSelector`

**classmethod from\_text**(*text*, *start=0*, *end=None*)

Make position selector including the text strings “start” and “end” within “text”.

**Return type** `TextPositionSelector`

**range**()

Get the range of the text.

**Return type** `Range`

**rangeset**()

Get the range set of the text.

**Return type** `RangeSet`

**select\_text**(*text*)

Get the quotation from text identified by start and end positions.



**Return type** `str`

**classmethod** `start_less_than_end(v, values)`

Verify start position is before the end position.

**Returns** the end position, which after the start position

**classmethod** `start_not_negative(v)`

Verify start position is not negative.

**Return type** `bool`

**Returns** the start position, which is not negative

**subtract\_integer(value)**

Reduce self's startpoint and endpoint by an integer.

**Return type** `TextPositionSelector`

**unique\_quote\_selector(text)**

Add text to prefix and suffix as needed to make selector unique in the source text.

**Parameters** `text (str)` – the passage where an exact quotation needs to be located

**Return type** `TextQuoteSelector`

**verify\_text\_positions(text)**

Verify that selector's text positions exist in text.

**Return type** `None`

**class** `anchorpoint.textselectors.TextPositionSet(**data)`

A set of `TextPositionSelectors`.

**\_\_add\_\_**(value)

Increase all startpoints and endpoints by the given amount.

**Parameters** `value (Union[int, TextPositionSelector, TextPositionSet])` – selector for another text interval, or integer to add to every start and end value in self's position selectors

**Return type** `TextPositionSet`

**Returns** a selector reflecting the combined range if possible, otherwise `None`

**\_\_ge\_\_**(other)

Test if self's rangeset includes all of other's rangeset.

**Return type** `bool`

**\_\_gt\_\_**(other)

Test if self's rangeset includes all of other's rangeset, but is not identical.

**Return type** `bool`

**\_\_hash\_\_** = `None`

**\_\_str\_\_**()

Return `str(self)`.

**\_\_sub\_\_**(value)

Decrease all startpoints and endpoints by the given amount.

**Return type** `TextPositionSet`

**add\_margin**(text, margin\_width=3, margin\_characters=',.\'";[>()')

Expand selected position selectors to include margin of punctuation.

This can cause multiple selections to be merged into a single one.

Ignores quote selectors.

#### Parameters

- **text** (*str*) – The text that passages are selected from
- **margin\_width** (*int*) – The width of the margin to add
- **margin\_characters** (*str*) – The characters to include in the margin

**Return type** *TextPositionSet*

**Returns** A new *TextPositionSet* with the margin added

```
>>> from anchorpoint.textselectors import TextPositionSetFactory
>>> text = "I predict that the grass is wet. (It rained.)"
>>> factory = TextPositionSetFactory(text=text)
>>> selectors = [TextQuoteSelector(exact="the grass is wet"),
↳TextQuoteSelector(exact="it rained")]
>>> position_set = factory.from_selection(selection=selectors)
>>> len(position_set.ranges())
2
>>> new_position_set = position_set.add_margin(text=text)
>>> len(new_position_set.ranges())
1
>>> new_position_set.ranges()[0].start
15
>>> new_position_set.ranges()[0].end
43
```

#### **as\_quotes**(*text*)

Copy self's quote and position selectors, converting all position selectors to quote selectors.

**Return type** *List[TextQuoteSelector]*

#### **as\_string**(*text*)

Return a string representing the selected parts of *text*.

```
>>> selectors = [TextPositionSelector(start=5, end=10)]
>>> selector_set = TextPositionSet(positions=selectors)
>>> sequence = selector_set.as_text_sequence("Some text.")
>>> selector_set.as_string("Some text.")
'...text.'
```

**Return type** *str*

#### **as\_text\_sequence**(*text, include\_nones=True*)

List the phrases in a text passage selected by this *TextPositionSet*.

#### Parameters

- **passage** – A passage to select text from
- **include\_nones** (*bool*) – Whether the list of phrases should include *None* to indicate a block of unselected text

**Return type** *TextSequence*

**Returns** A *TextSequence* of the phrases in the text

```

>>> selectors = [TextPositionSelector(start=5, end=10)]
>>> selector_set = TextPositionSet(positions=selectors)
>>> selector_set.as_text_sequence("Some text.")
TextSequence([None, TextPassage("text.")])

```

**convert\_quotes\_to\_positions**(*text*)

Return new TextPositionSet with all quotes replaced by their positions in the given text.

**Return type** ForwardRef

**classmethod from\_quotes**(*selection*)

Construct TextPositionSet from string or TextQuoteSelectors.

If a string is used, it will be converted to a *TextQuoteSelector* with no prefix or suffix.

**Return type** TextPositionSet

**classmethod from\_ranges**(*ranges*)

Make new class instance from Range objects from python-ranges library.

**Return type** TextPositionSet

**merge\_rangeset**(*rangeset*)

Merge another RangeSet into this one, returning a new TextPositionSet.

**Parameters** *rangeset* (RangeSet) – the RangeSet to merge

**Return type** ForwardRef

**Returns** a new TextPositionSet representing the combined ranges

**classmethod order\_of\_selectors**(*v*)

Ensure that selectors are in order.

**positions\_as\_quotes**(*text*)

Copy self's position selectors, converted to quote selectors.

**Return type** List[TextQuoteSelector]

**positions\_of\_quote\_selectors**(*text*)

Convert self's quote selectors to position selectors for a given text.

**Return type** List[TextPositionSelector]

**classmethod quote\_selectors\_are\_in\_list**(*selectors*)

Put single selector in list and convert strings to selectors.

**quotes\_rangeset**(*text*)

Get ranges where these quotes appear in the provided text.

**Return type** RangeSet

**ranges**()

Get positions as Range objects from python-ranges library.

**Return type** List[Range]

**rangeset**()

Convert positions into python-ranges Rangeset.

**Return type** RangeSet

**select\_text**(*text*, *margin\_width*=3, *margin\_characters*='.\' ;[>()')

Return the selected text from *text*.

**Parameters**

- **text** (*str*) – The text that passages are selected from
- **margin\_width** (*int*) – The width of the margin to add
- **margin\_characters** (*str*) – The characters to include in the margin

**Return type** *str***Returns** The selected text

```
>>> from anchorpoint.textselectors import TextPositionSetFactory
>>> text = "I predict that the grass is wet. (It rained.)"
>>> factory = TextPositionSetFactory(text=text)
>>> selectors = [TextQuoteSelector(exact="the grass is wet"),
↳ TextQuoteSelector(exact="it rained")]
>>> position_set = factory.from_selection(selection=selectors)
>>> position_set.select_text(text=text)
'...the grass is wet. (It rained...'
```

**classmethod** `selectors_are_in_list`(*selectors*)

Put single selector in list.

**class** `anchorpoint.textselectors.TextPositionSetFactory`(*text*)Factory for constructing *TextPositionSet* from text passages and various kinds of selector.**\_\_init\_\_**(*text*)

Store text passage that will be used to generate text selections.

**\_\_weakref\_\_**

list of weak references to the object (if defined)

**from\_bool**(*selection*)

Select either the whole passage or none of it.

**Return type** *TextPositionSet***from\_exact\_strings**(*selection*)Construct *TextPositionSet* from a sequence of strings representing exact quotations.First converts the sequence to *TextQuoteSelectors*, and then to *TextPositionSelectors*.**Return type** *TextPositionSet***from\_quote\_selectors**(*quotes*)Construct *TextPositionSet* from a sequence of *TextQuoteSelectors*.**Return type** *TextPositionSet***from\_selection**(*selection*)Construct *TextPositionSet* for a provided text passage, from any type of selector.**Return type** *TextPositionSet***from\_selection\_sequence**(*selections*)Construct *TextPositionSet* from one or more of: strings, Quote Selectors, and Position Selectors.First converts strings to *TextQuoteSelectors*, and then to *TextPositionSelectors*.**Return type** *TextPositionSet***class** `anchorpoint.textselectors.TextQuoteSelector`(*\*\*data*)

Describes a textual segment by quoting it, or passages before or after it.

Based on the Web Annotation Data Model [Text Quote Selector](#) standard

### Parameters

- **exact** – a copy of the text which is being selected
- **prefix** – a snippet of text that occurs immediately before the text which is being selected.
- **suffix** – the snippet of text that occurs immediately after the text which is being selected.

`__hash__ = None`

**as\_position**(*text*)

Get the interval where the selected quote appears in “text”.

**Parameters** **text** (*str*) – the passage where an exact quotation needs to be located

**Return type** *TextPositionSelector*

**Returns** the position selector for the location of the exact quotation

**as\_unique\_position**(*text*)

Get the interval where the selected quote appears in “text”.

**Parameters** **text** (*str*) – the passage where an exact quotation needs to be located

**Return type** *TextPositionSelector*

**Returns** the position selector for the location of the exact quotation

**find\_match**(*text*)

Get the first match for the selector within a string.

**Parameters** **text** (*str*) – text to search for a match to the selector

**Return type** *Optional*[*Match*]

**Returns** a regular expression match, or None

```
>>> text = "process, system, method of operation, concept, principle"
>>> selector = TextQuoteSelector(exact="method of operation")
>>> selector.find_match(text)
<re.Match object; span=(17, 36), match='method of operation'>
```

**classmethod from\_text**(*text*)

Create a selector from a text string.

“prefix” and “suffix” fields may be created by separating part of the text with a pipe character (“|”).

**Parameters** **text** (*str*) – the passage where an exact quotation needs to be located

**Return type** *TextQuoteSelector*

**Returns** a selector for the location of the exact quotation

```
>>> text = "process, system,|method of operation|, concept, principle"
>>> selector = TextQuoteSelector.from_text(text)
>>> selector.prefix
'process, system,'
>>> selector.exact
'method of operation'
>>> selector.suffix
', concept, principle'
```

**is\_unique\_in**(*text*)

Test if selector refers to exactly one passage in text.

**Parameters** **text** (**str**) – the passage where an exact quotation needs to be located

**Return type** **bool**

**Returns** whether the passage appears exactly once

**classmethod no\_none\_for\_prefix**(*value*)

Ensure that 'prefix', 'exact', and 'suffix' are not None.

**passage\_regex**()

Get regex to identify the selected text.

**prefix\_regex**()

Get regex for the text before any whitespace and the selection.

**rebuild\_from\_text**(*text*)

Make new selector with the "exact" value found in a given text.

Used for building a complete selector when exact has not been specified.

**Parameters** **text** (**str**) – the passage where an exact quotation needs to be located

**Return type** **Optional**[**TextQuoteSelector**]

**Returns** a new selector with the "exact" value found in the provided text

**select\_text**(*text*)

Get the passage matching the selector, minus any whitespace at ends.

**Parameters** **text** (**str**) – the passage where an exact quotation needs to be located.

**Return type** **Optional**[**str**]

**Returns** the passage between prefix and suffix in text.

```
>>> text = "process, system, method of operation, concept, principle"
>>> selector = TextQuoteSelector(prefix="method of operation,")
>>> selector.select_text(text)
'concept, principle'
```

**static split\_anchor\_text**(*text*)

Break up shorthand text selector format into three fields.

Tries to break up the string into prefix, exact, and suffix, by splitting on exactly two pipe characters.

**Parameters** **text** (**str**) – a string or dict representing a text passage

**Return type** **Tuple**[**str**, ...]

**Returns** a tuple of the three values

**suffix\_regex**()

Get regex for the text following the selection and any whitespace.

**exception anchorpoint.textselectors.TextSelectionError**

Exception for failing to select text as described by user.

**\_\_weakref\_\_**

list of weak references to the object (if defined)

## 3.2 Text Sequences

**class** anchorpoint.textsequences.**TextPassage**(*text*)

A contiguous passage of text.

Can be used to compare passages while disregarding end punctuation.

**means**(*other*)

Test if passages have the same text, disregarding end punctuation.

**Parameters** *other* (Optional[TextPassage]) – the other passage to test against

**Return type** bool

**Returns** True if the two passages have the same text, False otherwise

```
>>> TextPassage("Hello, world.").means(TextPassage("Hello, world"))
True
>>> TextPassage("Hello world").means(TextPassage("Hello, world"))
False
```

**class** anchorpoint.textsequences.**TextSequence**(*passages=None*)

Sequential passages of text that need not be consecutive.

Unlike a [Legislice Enactment](#), a TextSequence does not preserve the tree structure of the quoted document.

**\_\_add\_\_**(*other*)

Combine TextSequences by merging their selected *TextPassages*.

**Return type** TextSequence

**\_\_ge\_\_**(*other*)

Return self>=value.

**\_\_gt\_\_**(*other*)

Return self>value.

**\_\_init\_\_**(*passages=None*)

Make new TextSequence from *TextPassage* list.

**Parameters** *passages* (Optional[List[Optional[TextPassage]])] – the text passages included in the TextSequence, which should be chosen to express a coherent idea. “None”s in the sequence represent spans of text that exist in the source document, but that haven’t been chosen to be part of the TextSequence.

**\_\_repr\_\_**()

Return repr(self).

**\_\_str\_\_**()

Return str(self).

**\_\_weakref\_\_**

list of weak references to the object (if defined)

**means**(*other*)

Test if all the passages in self and other correspond with each other.

**Return type** bool

**strip**()

Remove symbols representing missing text from the beginning and end.

**Return type** TextSequence





## INDICES AND TABLES

- genindex
- search



## PYTHON MODULE INDEX

### a

`anchorpoint.textselectors`, 11



## Symbols

- `__add__()` (*anchorpoint.textselectors.TextPositionSelector* method), 11
  - `__add__()` (*anchorpoint.textselectors.TextPositionSet* method), 13
  - `__add__()` (*anchorpoint.textsequences.TextSequence* method), 19
  - `__and__()` (*anchorpoint.textselectors.TextPositionSelector* method), 11
  - `__ge__()` (*anchorpoint.textselectors.TextPositionSelector* method), 11
  - `__ge__()` (*anchorpoint.textselectors.TextPositionSet* method), 13
  - `__ge__()` (*anchorpoint.textsequences.TextSequence* method), 19
  - `__gt__()` (*anchorpoint.textselectors.TextPositionSelector* method), 11
  - `__gt__()` (*anchorpoint.textselectors.TextPositionSet* method), 13
  - `__gt__()` (*anchorpoint.textsequences.TextSequence* method), 19
  - `__hash__` (*anchorpoint.textselectors.TextPositionSelector* attribute), 12
  - `__hash__` (*anchorpoint.textselectors.TextPositionSet* attribute), 13
  - `__hash__` (*anchorpoint.textselectors.TextQuoteSelector* attribute), 17
  - `__init__()` (*anchorpoint.textselectors.TextPositionSetFactory* method), 16
  - `__init__()` (*anchorpoint.textsequences.TextSequence* method), 19
  - `__or__()` (*anchorpoint.textselectors.TextPositionSelector* method), 12
  - `__repr__()` (*anchorpoint.textsequences.TextSequence* method), 19
  - `__str__()` (*anchorpoint.textselectors.TextPositionSet* method), 13
  - `__str__()` (*anchorpoint.textsequences.TextSequence* method), 19
  - `__sub__()` (*anchorpoint.textselectors.TextPositionSet* method), 13
  - `__weakref__` (*anchorpoint.textselectors.TextPositionSetFactory* attribute), 16
  - `__weakref__` (*anchorpoint.textselectors.TextSelectionError* attribute), 18
  - `__weakref__` (*anchorpoint.textsequences.TextSequence* attribute), 19
- ## A
- `add_margin()` (*anchorpoint.textselectors.TextPositionSet* method), 13
  - `anchorpoint.textselectors` module, 11
  - `as_position()` (*anchorpoint.textselectors.TextQuoteSelector* method), 17
  - `as_quote()` (*anchorpoint.textselectors.TextPositionSelector* method), 12
  - `as_quotes()` (*anchorpoint.textselectors.TextPositionSet* method), 14
  - `as_string()` (*anchorpoint.textselectors.TextPositionSet* method), 14
  - `as_text_sequence()` (*anchorpoint.textselectors.TextPositionSet* method), 14
  - `as_unique_position()` (*anchorpoint.textselectors.TextQuoteSelector* method), 17
- ## C
- `combine()` (*anchorpoint.textselectors.TextPositionSelector* method), 12
  - `convert_quotes_to_positions()` (*anchorpoint.textselectors.TextPositionSet* method), 15
- ## D
- `difference()` (*anchorpoint.textselectors.TextPositionSelector* method), 12

## F

`find_match()` (*anchorpoint.textselectors.TextQuoteSelector* method), 17

`from_bool()` (*anchorpoint.textselectors.TextPositionSetFactory* method), 16

`from_exact_strings()` (*anchorpoint.textselectors.TextPositionSetFactory* method), 16

`from_quote_selectors()` (*anchorpoint.textselectors.TextPositionSetFactory* method), 16

`from_quotes()` (*anchorpoint.textselectors.TextPositionSet* class method), 15

`from_range()` (*anchorpoint.textselectors.TextPositionSelector* class method), 12

`from_ranges()` (*anchorpoint.textselectors.TextPositionSet* class method), 15

`from_selection()` (*anchorpoint.textselectors.TextPositionSetFactory* method), 16

`from_selection_sequence()` (*anchorpoint.textselectors.TextPositionSetFactory* method), 16

`from_text()` (*anchorpoint.textselectors.TextPositionSelector* class method), 12

`from_text()` (*anchorpoint.textselectors.TextQuoteSelector* class method), 17

## I

`is_unique_in()` (*anchorpoint.textselectors.TextQuoteSelector* method), 17

## M

`means()` (*anchorpoint.textsequences.TextPassage* method), 19

`means()` (*anchorpoint.textsequences.TextSequence* method), 19

`merge_rangeset()` (*anchorpoint.textselectors.TextPositionSet* method), 15

module  
    *anchorpoint.textselectors*, 11

## N

`no_none_for_prefix()` (*anchorpoint.textselectors.TextQuoteSelector* class

method), 18

## O

`order_of_selectors()` (*anchorpoint.textselectors.TextPositionSet* class method), 15

## P

`passage_regex()` (*anchorpoint.textselectors.TextQuoteSelector* method), 18

`positions_as_quotes()` (*anchorpoint.textselectors.TextPositionSet* method), 15

`positions_of_quote_selectors()` (*anchorpoint.textselectors.TextPositionSet* method), 15

`prefix_regex()` (*anchorpoint.textselectors.TextQuoteSelector* method), 18

## Q

`quote_selectors_are_in_list()` (*anchorpoint.textselectors.TextPositionSet* class method), 15

`quotes_rangeset()` (*anchorpoint.textselectors.TextPositionSet* method), 15

## R

`range()` (*anchorpoint.textselectors.TextPositionSelector* method), 12

`ranges()` (*anchorpoint.textselectors.TextPositionSet* method), 15

`rangeset()` (*anchorpoint.textselectors.TextPositionSelector* method), 12

`rangeset()` (*anchorpoint.textselectors.TextPositionSet* method), 15

`rebuild_from_text()` (*anchorpoint.textselectors.TextQuoteSelector* method), 18

## S

`select_text()` (*anchorpoint.textselectors.TextPositionSelector* method), 12

`select_text()` (*anchorpoint.textselectors.TextPositionSet* method), 15

`select_text()` (*anchorpoint.textselectors.TextQuoteSelector* method), 18

`selectors_are_in_list()` (*anchorpoint.textselectors.TextPositionSet* class method), 16

`split_anchor_text()` (*anchorpoint.textselectors.TextQuoteSelector* static method), 18

`start_less_than_end()` (*anchorpoint.textselectors.TextPositionSelector* class method), 13

`start_not_negative()` (*anchorpoint.textselectors.TextPositionSelector* class method), 13

`strip()` (*anchorpoint.textsequences.TextSequence* method), 19

`subtract_integer()` (*anchorpoint.textselectors.TextPositionSelector* method), 13

`suffix_regex()` (*anchorpoint.textselectors.TextQuoteSelector* method), 18

## T

`TextPassage` (*class in anchorpoint.textsequences*), 19

`TextPositionSelector` (*class in anchorpoint.textselectors*), 11

`TextPositionSet` (*class in anchorpoint.textselectors*), 13

`TextPositionSetFactory` (*class in anchorpoint.textselectors*), 16

`TextQuoteSelector` (*class in anchorpoint.textselectors*), 16

`TextSelectionError`, 18

`TextSequence` (*class in anchorpoint.textsequences*), 19

## U

`unique_quote_selector()` (*anchorpoint.textselectors.TextPositionSelector* method), 13

## V

`verify_text_positions()` (*anchorpoint.textselectors.TextPositionSelector* method), 13