ElementHandle

- extends: JSHandle

ElementHandle represents an in-page DOM element. ElementHandles can be created with the page.query_selector() method.

> **DISCOURAGED**
>
> The use of ElementHandle is discouraged, use Locator objects and web-first assertions instead.

Sync Async

```
href_element = page.query_selector("a")
href_element.click()
```

ElementHandle prevents DOM element from garbage collection unless the handle is disposed with js_handle.dispose(). ElementHandles are auto-disposed when their origin frame gets navigated.

ElementHandle instances can be used as an argument in page.eval_on_selector() and page.evaluate() methods.

The difference between the Locator and ElementHandle is that the ElementHandle points to a particular element, while Locator captures the logic of how to retrieve an element.

In the example below, handle points to a particular DOM element on page. If that element changes text or is used by React to render an entirely different component, handle is still pointing to that very DOM element. This can lead to unexpected behaviors.

Sync Async

```
handle = page.query_selector("text=Submit")
handle.hover()
```

```
handle.click()
```

With the locator, every time the `element` is used, up-to-date DOM element is located in the page using the selector. So in the snippet below, underlying DOM element is going to be located twice.

Sync **Async**

```
locator = page.get_by_text("Submit")
locator.hover()
locator.click()
```

Methods

bounding_box

Added in: v1.8

This method returns the bounding box of the element, or `null` if the element is not visible. The bounding box is calculated relative to the main frame viewport - which is usually the same as the browser window.

Scrolling affects the returned bounding box, similarly to Element.getBoundingClientRect. That means `x` and/or `y` may be negative.

Elements from child frames return the bounding box relative to the main frame, unlike the Element.getBoundingClientRect.

Assuming the page is static, it is safe to use bounding box coordinates to perform input. For example, the following snippet should click the center of the element.

Usage

Sync **Async**

```
box = element_handle.bounding_box()
page.mouse.click(box["x"] + box["width"] / 2, box["y"] + box["height"] / 2)
```

Returns

- NoneType|Dict
  - `x` float

    the x coordinate of the element in pixels.
  - `y` float

    the y coordinate of the element in pixels.
  - `width` float

    the width of the element in pixels.
  - `height` float

    the height of the element in pixels.

check

Added in: v1.8

This method checks the element by performing the following steps:

1. Ensure that element is a checkbox or a radio input. If not, this method throws. If the element is already checked, this method returns immediately.
2. Wait for actionability checks on the element, unless `force` option is set.
3. Scroll the element into view if needed.
4. Use page.mouse to click in the center of the element.
5. Wait for initiated navigations to either succeed or fail, unless `no_wait_after` option is set.
6. Ensure that the element is now checked. If not, this method throws.

If the element is detached from the DOM at any moment during the action, this method throws.

When all steps combined have not finished during the specified `timeout`, this method throws a TimeoutError. Passing zero timeout disables this.

Usage

```
element_handle.check()
element_handle.check(**kwargs)
```

Arguments

- `force` bool *(optional)*

  Whether to bypass the actionability checks. Defaults to `false`.

- `no_wait_after` bool *(optional)*

  Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `position` Dict *(optional)* Added in: v1.11
  - `x` float
  - `y` float

  A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

- `trial` bool *(optional)* Added in: v1.11

  When set, this method only performs the actionability checks and skips the action. Defaults to `false`. Useful to wait until the element is ready for the action without performing it.

click

Added in: v1.8

This method clicks the element by performing the following steps:

1. Wait for actionability checks on the element, unless `force` option is set.
2. Scroll the element into view if needed.

3. Use page.mouse to click in the center of the element, or the specified `position`.
4. Wait for initiated navigations to either succeed or fail, unless `no_wait_after` option is set.

If the element is detached from the DOM at any moment during the action, this method throws.

When all steps combined have not finished during the specified `timeout`, this method throws a TimeoutError. Passing zero timeout disables this.

Usage

```
element_handle.click()
element_handle.click(**kwargs)
```

Arguments

- `button` "left"|"right"|"middle" *(optional)*

  Defaults to `left`.

- `click_count` int *(optional)*

  defaults to 1. See UIEvent.detail.

- `delay` float *(optional)*

  Time to wait between `mousedown` and `mouseup` in milliseconds. Defaults to 0.

- `force` bool *(optional)*

  Whether to bypass the actionability checks. Defaults to `false`.

- `modifiers` List["Alt"|"Control"|"Meta"|"Shift"] *(optional)*

  Modifier keys to press. Ensures that only these modifiers are pressed during the operation, and then restores current modifiers back. If not specified, currently pressed modifiers are used.

- `no_wait_after` bool *(optional)*

  Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `position` Dict *(optional)*
  - `x` float
  - `y` float

  A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

- `trial` bool *(optional)* Added in: v1.11

  When set, this method only performs the actionability checks and skips the action. Defaults to `false`. Useful to wait until the element is ready for the action without performing it.

content_frame

Added in: v1.8

Returns the content frame for element handles referencing iframe nodes, or `null` otherwise

Usage

```
element_handle.content_frame()
```

Returns

- NoneType|Frame

dblclick

Added in: v1.8

This method double clicks the element by performing the following steps:

1. Wait for actionability checks on the element, unless `force` option is set.

2. Scroll the element into view if needed.
3. Use page.mouse to double click in the center of the element, or the specified `position`.
4. Wait for initiated navigations to either succeed or fail, unless `no_wait_after` option is set. Note that if the first click of the `dblclick()` triggers a navigation event, this method will throw.

If the element is detached from the DOM at any moment during the action, this method throws.

When all steps combined have not finished during the specified `timeout`, this method throws a TimeoutError. Passing zero timeout disables this.

> **NOTE**
> `elementHandle.dblclick()` dispatches two `click` events and a single `dblclick` event.

Usage

```
element_handle.dblclick()
element_handle.dblclick(**kwargs)
```

Arguments

- `button` "left"|"right"|"middle" *(optional)*

  Defaults to `left`.

- `delay` float *(optional)*

  Time to wait between `mousedown` and `mouseup` in milliseconds. Defaults to 0.

- `force` bool *(optional)*

  Whether to bypass the actionability checks. Defaults to `false`.

- `modifiers` List["Alt"|"Control"|"Meta"|"Shift"] *(optional)*

  Modifier keys to press. Ensures that only these modifiers are pressed during the operation, and then restores current modifiers back. If not specified, currently pressed modifiers are used.

- `no_wait_after` bool *(optional)*

Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `position` Dict *(optional)*

  - `x` float
  - `y` float

  A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

- `trial` bool *(optional)* Added in: v1.11

  When set, this method only performs the actionability checks and skips the action. Defaults to `false`. Useful to wait until the element is ready for the action without performing it.

dispatch_event

Added in: v1.8

The snippet below dispatches the `click` event on the element. Regardless of the visibility state of the element, `click` is dispatched. This is equivalent to calling element.click().

Usage

Sync **Async**

```
element_handle.dispatch_event("click")
```

Under the hood, it creates an instance of an event based on the given `type`, initializes it with `event_init` properties and dispatches it on the element. Events are `composed`, `cancelable` and bubble by default.

Since `event_init` is event-specific, please refer to the events documentation for the lists of initial properties:

- DragEvent
- FocusEvent
- KeyboardEvent
- MouseEvent
- PointerEvent
- TouchEvent
- Event

You can also specify `JSHandle` as the property value if you want live objects to be passed into the event:

Sync **Async**

```
# note you can only create data_transfer in chromium and firefox
data_transfer = page.evaluate_handle("new DataTransfer()")
element_handle.dispatch_event("#source", "dragstart", {"dataTransfer":
data_transfer})
```

Arguments

- `type` str

  DOM event type: `"click"`, `"dragstart"`, etc.

- `event_init` EvaluationArgument *(optional)*

  Optional event-specific initialization properties.

eval_on_selector

Added in: v1.9

Returns the return value of `expression`.

The method finds an element matching the specified selector in the `ElementHandle`s subtree and passes it as a first argument to `expression`. If no elements match the selector, the method

throws an error.

If `expression` returns a Promise, then element_handle.eval_on_selector() would wait for the promise to resolve and return its value.

Usage

Sync **Async**

```
tweet_handle = page.query_selector(".tweet")
assert tweet_handle.eval_on_selector(".like", "node => node.innerText") == "100"
assert tweet_handle.eval_on_selector(".retweets", "node => node.innerText") ==
"10"
```

Arguments

- `selector` str

  A selector to query for.

- `expression` str

  JavaScript expression to be evaluated in the browser context. If the expression evaluates to a function, the function is automatically invoked.

- `arg` EvaluationArgument *(optional)*

  Optional argument to pass to `expression`.

Returns

- Serializable

eval_on_selector_all

Added in: v1.9

Returns the return value of `expression`.

The method finds all elements matching the specified selector in the `ElementHandle`'s subtree and passes an array of matched elements as a first argument to `expression`.

If `expression` returns a Promise, then element_handle.eval_on_selector_all() would wait for the promise to resolve and return its value.

Usage

```
<div class="feed">
  <div class="tweet">Hello!</div>
  <div class="tweet">Hi!</div>
</div>
```

Sync **Async**

```
feed_handle = page.query_selector(".feed")
assert feed_handle.eval_on_selector_all(".tweet", "nodes => nodes.map(n =>
n.innerText)") == ["hello!", "hi!"]
```

Arguments

- `selector` str

  A selector to query for.

- `expression` str

  JavaScript expression to be evaluated in the browser context. If the expression evaluates to a function, the function is automatically invoked.

- `arg` EvaluationArgument *(optional)*

  Optional argument to pass to `expression`.

Returns

- Serializable

fill

Added in: v1.8

This method waits for actionability checks, focuses the element, fills it and triggers an `input` event after filling. Note that you can pass an empty string to clear the input field.

If the target element is not an `<input>`, `<textarea>` or `[contenteditable]` element, this method throws an error. However, if the element is inside the `<label>` element that has an associated control, the control will be filled instead.

To send fine-grained keyboard events, use locator.press_sequentially().

Usage

```
element_handle.fill(value)
element_handle.fill(value, **kwargs)
```

Arguments

- `value` str

  Value to set for the `<input>`, `<textarea>` or `[contenteditable]` element.

- `force` bool *(optional)* Added in: v1.13

  Whether to bypass the actionability checks. Defaults to `false`.

- `no_wait_after` bool *(optional)*

  Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

focus

Added in: v1.8

Calls focus on the element.

Usage

```
element_handle.focus()
```

get_attribute

Added in: v1.8

Returns element attribute value.

Usage

```
element_handle.get_attribute(name)
```

Arguments

- `name` str

  Attribute name to get the value for.

Returns

- NoneType|str

hover

Added in: v1.8

This method hovers over the element by performing the following steps:

1. Wait for actionability checks on the element, unless `force` option is set.
2. Scroll the element into view if needed.
3. Use page.mouse to hover over the center of the element, or the specified `position`.
4. Wait for initiated navigations to either succeed or fail, unless `noWaitAfter` option is set.

If the element is detached from the DOM at any moment during the action, this method throws.

When all steps combined have not finished during the specified `timeout`, this method throws a TimeoutError. Passing zero timeout disables this.

Usage

```
element_handle.hover()
element_handle.hover(**kwargs)
```

Arguments

- `force` bool *(optional)*

  Whether to bypass the actionability checks. Defaults to `false`.

- `modifiers` List["Alt"|"Control"|"Meta"|"Shift"] *(optional)*

  Modifier keys to press. Ensures that only these modifiers are pressed during the operation, and then restores current modifiers back. If not specified, currently pressed modifiers are used.

- `no_wait_after` bool *(optional)* Added in: v1.28

  Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `position` Dict *(optional)*

  - `x` float
  - `y` float

  A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

- `trial` bool *(optional)* Added in: v1.11

  When set, this method only performs the actionability checks and skips the action. Defaults to `false`. Useful to wait until the element is ready for the action without performing it.

inner_html

Added in: v1.8

Returns the `element.innerHTML`.

Usage

```
element_handle.inner_html()
```

Returns

- str

inner_text

Added in: v1.8

Returns the `element.innerText`.

Usage

```
element_handle.inner_text()
```

Returns

- str

input_value

Added in: v1.13

Returns `input.value` for the selected `<input>` or `<textarea>` or `<select>` element.

Throws for non-input elements. However, if the element is inside the `<label>` element that has an associated control, returns the value of the control.

Usage

```
element_handle.input_value()
```

```
element_handle.input_value(**kwargs)
```

Arguments

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

Returns

- str

is_checked

Added in: v1.8

Returns whether the element is checked. Throws if the element is not a checkbox or radio input.

Usage

```
element_handle.is_checked()
```

Returns

- bool

is_disabled

Added in: v1.8

Returns whether the element is disabled, the opposite of enabled.

Usage

```
element_handle.is_disabled()
```

Returns

- bool

is_editable

Added in: v1.8

Returns whether the element is editable.

Usage

```
element_handle.is_editable()
```

Returns

- bool

is_enabled

Added in: v1.8

Returns whether the element is enabled.

Usage

```
element_handle.is_enabled()
```

Returns

- bool

is_hidden

Added in: v1.8

Returns whether the element is hidden, the opposite of visible.

Usage

```
element_handle.is_hidden()
```

Returns

- bool

is_visible

Added in: v1.8

Returns whether the element is visible.

Usage

```
element_handle.is_visible()
```

Returns

- bool

owner_frame

Added in: v1.8

Returns the frame containing the given element.

Usage

```
element_handle.owner_frame()
```

Returns

- NoneType|Frame

press

Added in: v1.8

Focuses the element, and then uses keyboard.down() and keyboard.up().

`key` can specify the intended keyboardEvent.key value or a single character to generate the text for. A superset of the `key` values can be found here. Examples of the keys are:

`F1` - `F12`, `Digit0`- `Digit9`, `KeyA`- `KeyZ`, `Backquote`, `Minus`, `Equal`, `Backslash`, `Backspace`, `Tab`, `Delete`, `Escape`, `ArrowDown`, `End`, `Enter`, `Home`, `Insert`, `PageDown`, `PageUp`, `ArrowRight`, `ArrowUp`, etc.

Following modification shortcuts are also supported: `Shift`, `Control`, `Alt`, `Meta`, `ShiftLeft`.

Holding down `Shift` will type the text that corresponds to the `key` in the upper case.

If `key` is a single character, it is case-sensitive, so the values `a` and `A` will generate different respective texts.

Shortcuts such as `key: "Control+o"` or `key: "Control+Shift+T"` are supported as well. When specified with the modifier, modifier is pressed and being held while the subsequent key is being pressed.

Usage

```
element_handle.press(key)
element_handle.press(key, **kwargs)
```

Arguments

- `key` str

  Name of the key to press or a character to generate, such as `ArrowLeft` or `a`.

- `delay` float *(optional)*

  Time to wait between `keydown` and `keyup` in milliseconds. Defaults to 0.

- `no_wait_after` bool *(optional)*

  Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

query_selector

Added in: v1.9

The method finds an element matching the specified selector in the `ElementHandle`'s subtree. If no elements match the selector, returns `null`.

Usage

```
element_handle.query_selector(selector)
```

Arguments

- `selector` str

  A selector to query for.

Returns

- NoneType|ElementHandle

query_selector_all

Added in: v1.9

The method finds all elements matching the specified selector in the `ElementHandle`s subtree. If no elements match the selector, returns empty array.

Usage

```
element_handle.query_selector_all(selector)
```

Arguments

- `selector` str

  A selector to query for.

Returns

- List[ElementHandle]

screenshot

Added in: v1.8

This method captures a screenshot of the page, clipped to the size and position of this particular element. If the element is covered by other elements, it will not be actually visible on the screenshot. If the element is a scrollable container, only the currently scrolled content will be visible on the screenshot.

This method waits for the actionability checks, then scrolls element into view before taking a screenshot. If the element is detached from DOM, the method throws an error.

Returns the buffer with the captured screenshot.

Usage

```
element_handle.screenshot()
element_handle.screenshot(**kwargs)
```

Arguments

- `animations` "disabled"|"allow" *(optional)*

  When set to `"disabled"`, stops CSS animations, CSS transitions and Web Animations. Animations get different treatment depending on their duration:

  - finite animations are fast-forwarded to completion, so they'll fire `transitionend` event.
  - infinite animations are canceled to initial state, and then played over after the screenshot.

  Defaults to `"allow"` that leaves animations untouched.

- `caret` "hide"|"initial" *(optional)*

  When set to `"hide"`, screenshot will hide text caret. When set to `"initial"`, text caret behavior will not be changed. Defaults to `"hide"`.

- `mask` List[Locator] *(optional)*

  Specify locators that should be masked when the screenshot is taken. Masked elements will be overlaid with a pink box `#FF00FF` (customized by `mask_color`) that completely covers its bounding box.

- `mask_color` str *(optional)* Added in: v1.35

  Specify the color of the overlay box for masked elements, in CSS color format. Default color is pink `#FF00FF`.

- `omit_background` bool *(optional)*

  Hides default white background and allows capturing screenshots with transparency. Not applicable to `jpeg` images. Defaults to `false`.

- `path` Union[str, pathlib.Path] *(optional)*

  The file path to save the image to. The screenshot type will be inferred from file extension. If `path` is a relative path, then it is resolved relative to the current working directory. If no path is provided, the image won't be saved to the disk.

- `quality` int *(optional)*

  The quality of the image, between 0-100. Not applicable to `png` images.

- `scale` "css"|"device" *(optional)*

  When set to `"css"`, screenshot will have a single pixel per each css pixel on the page. For high-dpi devices, this will keep screenshots small. Using `"device"` option will produce a single pixel per each device pixel, so screenshots of high-dpi devices will be twice as large or even larger.

  Defaults to `"device"`.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

- `type` "png"|"jpeg" *(optional)*

  Specify screenshot type, defaults to `png`.

Returns

- bytes

scroll_into_view_if_needed

Added in: v1.8

This method waits for actionability checks, then tries to scroll element into view, unless it is completely visible as defined by IntersectionObserver's `ratio`.

Throws when `elementHandle` does not point to an element connected to a Document or a ShadowRoot.

Usage

```
element_handle.scroll_into_view_if_needed()
element_handle.scroll_into_view_if_needed(**kwargs)
```

Arguments

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

select_option

Added in: v1.8

This method waits for actionability checks, waits until all specified options are present in the `<select>` element and selects these options.

If the target element is not a `<select>` element, this method throws an error. However, if the element is inside the `<label>` element that has an associated control, the control will be used instead.

Returns the array of option values that have been successfully selected.

Triggers a `change` and `input` event once all the provided options have been selected.

Usage

Sync **Async**

```
# Single selection matching the value or label
handle.select_option("blue")
# single selection matching both the label
handle.select_option(label="blue")
# multiple selection
handle.select_option(value=["red", "green", "blue"])
```

Arguments

- `force` bool *(optional)* Added in: v1.13

  Whether to bypass the actionability checks. Defaults to `false`.

- `no_wait_after` bool *(optional)*

  Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

- `element` ElementHandle|List[ElementHandle] *(optional)*

  Option elements to select. Optional.

- `index` int|List[int] *(optional)*

  Options to select by index. Optional.

- `value` str|List[str] *(optional)*

  Options to select by value. If the `<select>` has the `multiple` attribute, all given options are selected, otherwise only the first option matching one of the passed options is selected. Optional.

- `label` str|List[str] *(optional)*

  Options to select by label. If the `<select>` has the `multiple` attribute, all given options are selected, otherwise only the first option matching one of the passed options is selected.

Optional.

Returns

- List[str]

select_text

Added in: v1.8

This method waits for actionability checks, then focuses the element and selects all its text content.

If the element is inside the `<label>` element that has an associated control, focuses and selects text in the control instead.

Usage

```
element_handle.select_text()
element_handle.select_text(**kwargs)
```

Arguments

- `force` bool *(optional)* Added in: v1.13

  Whether to bypass the actionability checks. Defaults to `false`.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

set_checked

Added in: v1.15

This method checks or unchecks an element by performing the following steps:

1. Ensure that element is a checkbox or a radio input. If not, this method throws.
2. If the element already has the right checked state, this method returns immediately.

3. Wait for actionability checks on the matched element, unless `force` option is set. If the element is detached during the checks, the whole action is retried.
4. Scroll the element into view if needed.
5. Use page.mouse to click in the center of the element.
6. Wait for initiated navigations to either succeed or fail, unless `no_wait_after` option is set.
7. Ensure that the element is now checked or unchecked. If not, this method throws.

When all steps combined have not finished during the specified `timeout`, this method throws a TimeoutError. Passing zero timeout disables this.

Usage

```
element_handle.set_checked(checked)
element_handle.set_checked(checked, **kwargs)
```

Arguments

- `checked` bool

  Whether to check or uncheck the checkbox.

- `force` bool *(optional)*

  Whether to bypass the actionability checks. Defaults to `false`.

- `no_wait_after` bool *(optional)*

  Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `position` Dict *(optional)*

  - `x` float
  - `y` float

  A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.

- `timeout` float *(optional)*

Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

- `trial` bool *(optional)*

  When set, this method only performs the actionability checks and skips the action. Defaults to `false`. Useful to wait until the element is ready for the action without performing it.

set_input_files

Added in: v1.8

Sets the value of the file input to these file paths or files. If some of the `filePaths` are relative paths, then they are resolved relative to the current working directory. For empty array, clears the selected files.

This method expects ElementHandle to point to an input element. However, if the element is inside the `<label>` element that has an associated control, targets the control instead.

Usage

```
element_handle.set_input_files(files)
element_handle.set_input_files(files, **kwargs)
```

Arguments

- `files` Union[str, pathlib.Path]|List[Union[str, pathlib.Path]]|Dict|List[Dict]
  - `name` str

    File name

  - `mimeType` str

    File type

  - `buffer` bytes

    File content

- `no_wait_after` bool *(optional)*

Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

tap

Added in: v1.8

This method taps the element by performing the following steps:

1. Wait for actionability checks on the element, unless `force` option is set.
2. Scroll the element into view if needed.
3. Use page.touchscreen to tap the center of the element, or the specified `position`.
4. Wait for initiated navigations to either succeed or fail, unless `no_wait_after` option is set.

If the element is detached from the DOM at any moment during the action, this method throws.

When all steps combined have not finished during the specified `timeout`, this method throws a TimeoutError. Passing zero timeout disables this.

> **NOTE**
>
> `elementHandle.tap()` requires that the `hasTouch` option of the browser context be set to true.

Usage

```
element_handle.tap()
element_handle.tap(**kwargs)
```

Arguments

- `force` bool *(optional)*

  Whether to bypass the actionability checks. Defaults to `false`.

- `modifiers` List["Alt"|"Control"|"Meta"|"Shift"] *(optional)*

  Modifier keys to press. Ensures that only these modifiers are pressed during the operation, and then restores current modifiers back. If not specified, currently pressed modifiers are used.

- `no_wait_after` bool *(optional)*

  Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `position` Dict *(optional)*

  - `x` float
  - `y` float

  A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

- `trial` bool *(optional)* Added in: v1.11

  When set, this method only performs the actionability checks and skips the action. Defaults to `false`. Useful to wait until the element is ready for the action without performing it.

text_content

Added in: v1.8

Returns the `node.textContent`.

Usage

```
element_handle.text_content()
```

Returns

- NoneType|str

uncheck

Added in: v1.8

This method checks the element by performing the following steps:

1. Ensure that element is a checkbox or a radio input. If not, this method throws. If the element is already unchecked, this method returns immediately.
2. Wait for actionability checks on the element, unless `force` option is set.
3. Scroll the element into view if needed.
4. Use page.mouse to click in the center of the element.
5. Wait for initiated navigations to either succeed or fail, unless `no_wait_after` option is set.
6. Ensure that the element is now unchecked. If not, this method throws.

If the element is detached from the DOM at any moment during the action, this method throws.

When all steps combined have not finished during the specified `timeout`, this method throws a TimeoutError. Passing zero timeout disables this.

Usage

```
element_handle.uncheck()
element_handle.uncheck(**kwargs)
```

Arguments

- `force` bool *(optional)*

  Whether to bypass the actionability checks. Defaults to `false`.

- `no_wait_after` bool *(optional)*

  Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `position` Dict *(optional)* Added in: v1.11

- `x` float
- `y` float

A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

- `trial` bool *(optional)* Added in: v1.11

  When set, this method only performs the actionability checks and skips the action. Defaults to `false`. Useful to wait until the element is ready for the action without performing it.

wait_for_element_state

Added in: v1.8

Returns when the element satisfies the `state`.

Depending on the `state` parameter, this method waits for one of the actionability checks to pass. This method throws when the element is detached while waiting, unless waiting for the `"hidden"` state.

- `"visible"` Wait until the element is visible.
- `"hidden"` Wait until the element is not visible or not attached. Note that waiting for hidden does not throw when the element detaches.
- `"stable"` Wait until the element is both visible and stable.
- `"enabled"` Wait until the element is enabled.
- `"disabled"` Wait until the element is not enabled.
- `"editable"` Wait until the element is editable.

If the element does not satisfy the condition for the `timeout` milliseconds, this method will throw.

Usage

```
element_handle.wait_for_element_state(state)
```

```
element_handle.wait_for_element_state(state, **kwargs)
```

Arguments

- `state` "visible"|"hidden"|"stable"|"enabled"|"disabled"|"editable"

  A state to wait for, see below for more details.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

wait_for_selector

Added in: v1.8

Returns element specified by selector when it satisfies `state` option. Returns `null` if waiting for `hidden` or `detached`.

Wait for the `selector` relative to the element handle to satisfy `state` option (either appear/disappear from dom, or become visible/hidden). If at the moment of calling the method `selector` already satisfies the condition, the method will return immediately. If the selector doesn't satisfy the condition for the `timeout` milliseconds, the function will throw.

Usage

Sync | Async

```
page.set_content("<div><span></span></div>")
div = page.query_selector("div")
# waiting for the "span" selector relative to the div.
span = div.wait_for_selector("span", state="attached")
```

> **NOTE**
>
> This method does not work across navigations, use page.wait_for_selector() instead.

Arguments

- `selector` str

  A selector to query for.

- `state` "attached"|"detached"|"visible"|"hidden" *(optional)*

  Defaults to `'visible'`. Can be either:

  - `'attached'` - wait for element to be present in DOM.
  - `'detached'` - wait for element to not be present in DOM.
  - `'visible'` - wait for element to have non-empty bounding box and no `visibility:hidden`. Note that element without any content or with `display:none` has an empty bounding box and is not considered visible.
  - `'hidden'` - wait for element to be either detached from DOM, or have an empty bounding box or `visibility:hidden`. This is opposite to the `'visible'` option.

- `strict` bool *(optional)* Added in: v1.15

  When true, the call requires selector to resolve to a single element. If given selector resolves to more than one element, the call throws an exception.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.

Returns

- NoneType|ElementHandle

Deprecated

type

Added in: v1.8

> **DEPRECATED**

> In most cases, you should use locator.fill() instead. You only need to press keys one by one if there is special keyboard handling on the page - in this case use locator.press_sequentially().

Focuses the element, and then sends a `keydown`, `keypress`/`input`, and `keyup` event for each character in the text.

To press a special key, like `Control` or `ArrowDown`, use element_handle.press().

Usage

Arguments

- `text` str

  A text to type into a focused element.

- `delay` float *(optional)*

  Time to wait between key presses in milliseconds. Defaults to 0.

- `no_wait_after` bool *(optional)*

  Actions that initiate navigations are waiting for these navigations to happen and for pages to start loading. You can opt out of waiting via setting this flag. You would only need this option in the exceptional cases such as navigating to inaccessible pages. Defaults to `false`.

- `timeout` float *(optional)*

  Maximum time in milliseconds. Defaults to `30000` (30 seconds). Pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_timeout() or page.set_default_timeout() methods.