Page

- extends: EventEmitter

Page provides methods to interact with a single tab in a Browser, or an extension background page in Chromium. One Browser instance might have multiple Page instances.

This example creates a page, navigates it to a URL, and then saves a screenshot:

Sync **Async**

```
from playwright.sync_api import sync_playwright, Playwright

def run(playwright: Playwright):
    webkit = playwright.webkit
    browser = webkit.launch()
    context = browser.new_context()
    page = context.new_page()
    page.goto("https://example.com")
    page.screenshot(path="screenshot.png")
    browser.close()

with sync_playwright() as playwright:
    run(playwright)
```

The Page class emits various events (described below) which can be handled using any of Node's native `EventEmitter` methods, such as `on`, `once` or `removeListener`.

This example logs a message for a single page `load` event:

```
page.once("load", lambda: print("page loaded!"))
```

To unsubscribe from events use the `removeListener` method:

```
def log_request(intercepted_request):
    print("a request was made:", intercepted_request.url)
page.on("request", log_request)
```

```
# sometime later...
page.remove_listener("request", log_request)
```

Methods

add_init_script

Added in: v1.8

Adds a script which would be evaluated in one of the following scenarios:

- Whenever the page is navigated.
- Whenever the child frame is attached or navigated. In this case, the script is evaluated in the context of the newly attached frame.

The script is evaluated after the document was created but before any of its scripts were run. This is useful to amend the JavaScript environment, e.g. to seed `Math.random`.

Usage

An example of overriding `Math.random` before the page loads:

```
// preload.js
Math.random = () => 42;
```

Sync | Async

```
# in your playwright script, assuming the preload.js file is in same directory
page.add_init_script(path="./preload.js")
```

> **NOTE**
>
> The order of evaluation of multiple scripts installed via browser_context.add_init_script() and page.add_init_script() is not defined.

Arguments

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

  Path to the JavaScript file. If `path` is a relative path, then it is resolved relative to the current working directory. Optional.

- `script` str *(optional)*

  Script to be evaluated in all pages in the browser context. Optional.

add_script_tag

Added in: v1.8

Adds a `<script>` tag into the page with the desired url or content. Returns the added tag when the script's onload fires or when the script content was injected into frame.

Usage

```
page.add_script_tag()
page.add_script_tag(**kwargs)
```

Arguments

- `content` str *(optional)*

  Raw JavaScript content to be injected into frame.

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

  Path to the JavaScript file to be injected into frame. If `path` is a relative path, then it is resolved relative to the current working directory.

- `type` str *(optional)*

  Script type. Use 'module' in order to load a Javascript ES6 module. See script for more details.

- `url` str *(optional)*

  URL of a script to be added.

Returns

- ElementHandle

add_style_tag

Added in: v1.8

Adds a `<link rel="stylesheet">` tag into the page with the desired url or a `<style type="text/css">` tag with the content. Returns the added tag when the stylesheet's onload fires or when the CSS content was injected into frame.

Usage

```
page.add_style_tag()
page.add_style_tag(**kwargs)
```

Arguments

- `content` str *(optional)*

  Raw CSS content to be injected into frame.

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

  Path to the CSS file to be injected into frame. If `path` is a relative path, then it is resolved relative to the current working directory.

- `url` str *(optional)*

  URL of the `<link>` tag.

Returns

- ElementHandle

bring_to_front

Added in: v1.8

Brings page to front (activates tab).

Usage

```
page.bring_to_front()
```

close

Added in: v1.8

If `run_before_unload` is `false`, does not run any unload handlers and waits for the page to be closed. If `run_before_unload` is `true` the method will run unload handlers, but will **not** wait for the page to close.

By default, `page.close()` **does not** run `beforeunload` handlers.

> **NOTE**
>
> if `run_before_unload` is passed as true, a `beforeunload` dialog might be summoned and should be handled manually via page.on("dialog") event.

Usage

```
page.close()
page.close(**kwargs)
```

Arguments

- `run_before_unload` bool *(optional)*

  Defaults to `false`. Whether to run the before unload page handlers.

content

Added in: v1.8

Gets the full HTML contents of the page, including the doctype.

Usage

```
page.content()
```

Returns

- str

drag_and_drop

Added in: v1.13

This method drags the source element to the target element. It will first move to the source element, perform a `mousedown`, then move to the target element and perform a `mouseup`.

Usage

Sync **Async**

```
page.drag_and_drop("#source", "#target")
# or specify exact positions relative to the top-left corners of the elements:
page.drag_and_drop(
  "#source",
  "#target",
  source_position={"x": 34, "y": 7},
  target_position={"x": 10, "y": 20}
)
```

Arguments

- `source` str

  A selector to search for an element to drag. If there are multiple elements satisfying the selector, the first will be used.

- `target` str

  A selector to search for an element to drop onto. If there are multiple elements satisfying the selector, the first will be used.

- `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`.

- `source_position` Dict *(optional)* Added in: v1.14

  - `x` float
  - `y` float

  Clicks on the source element at this point relative to the top-left corner of the element's padding box. If not specified, some visible point of the element is used.

- `strict` bool *(optional)* Added in: v1.14

  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.

- `target_position` Dict *(optional)* Added in: v1.14

  - `x` float
  - `y` float

  Drops on the target element at this point relative to the top-left corner of the element's padding box. If not specified, some visible point of the element is used.

- `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.

emulate_media

Added in: v1.8

This method changes the `CSS media type` through the `media` argument, and/or the `'prefers-colors-scheme'` media feature, using the `colorScheme` argument.

Usage

Sync Async

```
page.evaluate("matchMedia('screen').matches")
# → True
page.evaluate("matchMedia('print').matches")
# → False

page.emulate_media(media="print")
page.evaluate("matchMedia('screen').matches")
# → False
page.evaluate("matchMedia('print').matches")
# → True

page.emulate_media()
page.evaluate("matchMedia('screen').matches")
# → True
page.evaluate("matchMedia('print').matches")
# → False
```

Sync Async

```
page.emulate_media(color_scheme="dark")
page.evaluate("matchMedia('(prefers-color-scheme: dark)').matches")
# → True
page.evaluate("matchMedia('(prefers-color-scheme: light)').matches")
# → False
page.evaluate("matchMedia('(prefers-color-scheme: no-preference)').matches")
```

Arguments

- `color_scheme` "light"|"dark"|"no-preference"|"null" *(optional)* Added in: v1.9

  Emulates `'prefers-colors-scheme'` media feature, supported values are `'light'`, `'dark'`, `'no-preference'`. Passing `'Null'` disables color scheme emulation.

- `forced_colors` "active"|"none"|"null" *(optional)* Added in: v1.15

- `media` "screen"|"print"|"null" *(optional)* Added in: v1.9

Changes the CSS media type of the page. The only allowed values are `'Screen'`, `'Print'` and `'Null'`. Passing `'Null'` disables CSS media emulation.

- `reduced_motion` "reduce"|"no-preference"|"null" *(optional)* Added in: v1.12

  Emulates `'prefers-reduced-motion'` media feature, supported values are `'reduce'`, `'no-preference'`. Passing `null` disables reduced motion emulation.

evaluate

Added in: v1.8

Returns the value of the `expression` invocation.

If the function passed to the page.evaluate() returns a Promise, then page.evaluate() would wait for the promise to resolve and return its value.

If the function passed to the page.evaluate() returns a non-Serializable value, then page.evaluate() resolves to `undefined`. Playwright also supports transferring some additional values that are not serializable by `JSON`: `-0`, `NaN`, `Infinity`, `-Infinity`.

Usage

Passing argument to `expression`:

Sync **Async**

```
result = page.evaluate("([x, y]) => Promise.resolve(x * y)", [7, 8])
print(result) # prints "56"
```

A string can also be passed in instead of a function:

Sync **Async**

```
print(page.evaluate("1 + 2")) # prints "3"
x = 10
print(page.evaluate(f"1 + {x}")) # prints "11"
```

ElementHandle instances can be passed as an argument to the page.evaluate():

Sync Async

```
body_handle = page.evaluate("document.body")
html = page.evaluate("([body, suffix]) => body.innerHTML + suffix", [body_handle,
"hello"])
body_handle.dispose()
```

Arguments

- `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

evaluate_handle

Added in: v1.8

Returns the value of the `expression` invocation as a JSHandle.

The only difference between page.evaluate() and page.evaluate_handle() is that page.evaluate_handle() returns JSHandle.

If the function passed to the page.evaluate_handle() returns a Promise, then page.evaluate_handle() would wait for the promise to resolve and return its value.

Usage

Sync Async

```
a_window_handle = page.evaluate_handle("Promise.resolve(window)")
```

```
a_window_handle # handle for the window object.
```

A string can also be passed in instead of a function:

Sync Async

```
a_handle = page.evaluate_handle("document") # handle for the "document"
```

JSHandle instances can be passed as an argument to the page.evaluate_handle():

Sync Async

```
a_handle = page.evaluate_handle("document.body")
result_handle = page.evaluate_handle("body => body.innerHTML", a_handle)
print(result_handle.json_value())
result_handle.dispose()
```

Arguments

- `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

- JSHandle

expect_console_message

Added in: v1.9

Performs action and waits for a ConsoleMessage to be logged by in the page. If predicate is provided, it passes ConsoleMessage value into the `predicate` function and waits for

`predicate(message)` to return a truthy value. Will throw an error if the page is closed before the page.on("console") event is fired.

Usage

```
page.expect_console_message()
page.expect_console_message(**kwargs)
```

Arguments

- `predicate` Callable[ConsoleMessage]:bool *(optional)*

  Receives the ConsoleMessage object and resolves to truthy value when the waiting should resolve.

- `timeout` float *(optional)*

  Maximum time to wait for 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().

Returns

- EventContextManager[ConsoleMessage]

expect_download

Added in: v1.9

Performs action and waits for a new Download. If predicate is provided, it passes Download value into the `predicate` function and waits for `predicate(download)` to return a truthy value. Will throw an error if the page is closed before the download event is fired.

Usage

```
page.expect_download()
page.expect_download(**kwargs)
```

Arguments

- `predicate` Callable[Download]:bool *(optional)*

Receives the Download object and resolves to truthy value when the waiting should resolve.

- `timeout` float *(optional)*

  Maximum time to wait for 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().

Returns

- EventContextManager[Download]

expect_event

Added in: v1.8

Waits for event to fire and passes its value into the predicate function. Returns when the predicate returns truthy value. Will throw an error if the page is closed before the event is fired. Returns the event data value.

Usage

Sync | Async

```
with page.expect_event("framenavigated") as event_info:
    page.get_by_role("button")
frame = event_info.value
```

Arguments

- `event` str

  Event name, same one typically passed into `*.on(event)`.

- `predicate` Callable *(optional)*

  Receives the event data and resolves to truthy value when the waiting should resolve.

- `timeout` float *(optional)*

Maximum time to wait for 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().

Returns

- EventContextManager

expect_file_chooser

Added in: v1.9

Performs action and waits for a new FileChooser to be created. If predicate is provided, it passes FileChooser value into the `predicate` function and waits for `predicate(fileChooser)` to return a truthy value. Will throw an error if the page is closed before the file chooser is opened.

Usage

```
page.expect_file_chooser()
page.expect_file_chooser(**kwargs)
```

Arguments

- `predicate` Callable[FileChooser]:bool *(optional)*

  Receives the FileChooser object and resolves to truthy value when the waiting should resolve.

- `timeout` float *(optional)*

  Maximum time to wait for 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().

Returns

- EventContextManager[FileChooser]

expect_popup

Added in: v1.9

Performs action and waits for a popup Page. If predicate is provided, it passes [Popup] value into the `predicate` function and waits for `predicate(page)` to return a truthy value. Will throw an error if the page is closed before the popup event is fired.

Usage

```
page.expect_popup()
page.expect_popup(**kwargs)
```

Arguments

- `predicate` Callable[Page]:bool *(optional)*

  Receives the Page object and resolves to truthy value when the waiting should resolve.

- `timeout` float *(optional)*

  Maximum time to wait for 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().

Returns

- EventContextManager[Page]

expect_request

Added in: v1.8

Waits for the matching request and returns it. See waiting for event for more details about events.

Usage

Sync **Async**

```
with page.expect_request("http://example.com/resource") as first:
    page.get_by_text("trigger request").click()
first_request = first.value
```

```
# or with a lambda
with page.expect_request(lambda request: request.url == "http://example.com" and
request.method == "get") as second:
    page.get_by_text("trigger request").click()
second_request = second.value
```

Arguments

- `url_or_predicate` str|Pattern|Callable[Request]:bool

  Request URL string, regex or predicate receiving Request object. When a `base_url` via the context options was provided and the passed URL is a path, it gets merged via the `new URL()` constructor.

- `timeout` float *(optional)*

  Maximum wait time in milliseconds, defaults to 30 seconds, pass `0` to disable the timeout. The default value can be changed by using the page.set_default_timeout() method.

Returns

- EventContextManager[Request]

expect_request_finished

Added in: v1.12

Performs action and waits for a Request to finish loading. If predicate is provided, it passes Request value into the `predicate` function and waits for `predicate(request)` to return a truthy value. Will throw an error if the page is closed before the page.on("requestfinished") event is fired.

Usage

```
page.expect_request_finished()
page.expect_request_finished(**kwargs)
```

Arguments

- `predicate` Callable[Request]:bool *(optional)*

Receives the Request object and resolves to truthy value when the waiting should resolve.

- `timeout` float *(optional)*

  Maximum time to wait for 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().

Returns

- EventContextManager[Request]

expect_response

Added in: v1.8

Returns the matched response. See waiting for event for more details about events.

Usage

Sync | Async

```
with page.expect_response("https://example.com/resource") as response_info:
    page.get_by_text("trigger response").click()
response = response_info.value
return response.ok

# or with a lambda
with page.expect_response(lambda response: response.url == "https://example.com"
and response.status == 200) as response_info:
    page.get_by_text("trigger response").click()
response = response_info.value
return response.ok
```

Arguments

- `url_or_predicate` str|Pattern|Callable[Response]:bool

  Request URL string, regex or predicate receiving Response object. When a `base_url` via the context options was provided and the passed URL is a path, it gets merged via the `new URL()` constructor.

- `timeout` float *(optional)*

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

Returns

- EventContextManager[Response]

expect_websocket

Added in: v1.9

Performs action and waits for a new WebSocket. If predicate is provided, it passes WebSocket value into the `predicate` function and waits for `predicate(webSocket)` to return a truthy value. Will throw an error if the page is closed before the WebSocket event is fired.

Usage

```
page.expect_websocket()
page.expect_websocket(**kwargs)
```

Arguments

- `predicate` Callable[WebSocket]:bool *(optional)*

  Receives the WebSocket object and resolves to truthy value when the waiting should resolve.

- `timeout` float *(optional)*

  Maximum time to wait for 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().

Returns

- EventContextManager[WebSocket]

expect_worker

Added in: v1.9

Performs action and waits for a new Worker. If predicate is provided, it passes Worker value into the `predicate` function and waits for `predicate(worker)` to return a truthy value. Will throw an error if the page is closed before the worker event is fired.

Usage

```
page.expect_worker()
page.expect_worker(**kwargs)
```

Arguments

- `predicate` Callable[Worker]:bool *(optional)*

  Receives the Worker object and resolves to truthy value when the waiting should resolve.

- `timeout` float *(optional)*

  Maximum time to wait for 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().

Returns

- EventContextManager[Worker]

expose_binding

Added in: v1.8

The method adds a function called `name` on the `window` object of every frame in this page. When called, the function executes `callback` and returns a Promise which resolves to the return value of `callback`. If the `callback` returns a Promise, it will be awaited.

The first argument of the `callback` function contains information about the caller: `{ browserContext: BrowserContext, page: Page, frame: Frame }`.

See browser_context.expose_binding() for the context-wide version.

> **NOTE**

> Functions installed via page.expose_binding() survive navigations.

Usage

An example of exposing page URL to all frames in a page:

Sync Async

```
from playwright.sync_api import sync_playwright, Playwright

def run(playwright: Playwright):
    webkit = playwright.webkit
    browser = webkit.launch(headless=false)
    context = browser.new_context()
    page = context.new_page()
    page.expose_binding("pageURL", lambda source: source["page"].url)
    page.set_content("""
    <script>
      async function onClick() {
        document.querySelector('div').textContent = await window.pageURL();
      }
    </script>
    <button onclick="onClick()">Click me</button>
    <div></div>
    """)
    page.click("button")

with sync_playwright() as playwright:
    run(playwright)
```

An example of passing an element handle:

Sync Async

```
def print(source, element):
    print(element.text_content())

page.expose_binding("clicked", print, handle=true)
page.set_content("""
  <script>
    document.addEventListener('click', event => window.clicked(event.target));
```

```
  </script>
  <div>Click me</div>
  <div>Or click me</div>
""")
```

Arguments

- `name` str

  Name of the function on the window object.

- `callback` Callable

  Callback function that will be called in the Playwright's context.

- `handle` bool *(optional)*

  Whether to pass the argument as a handle, instead of passing by value. When passing a handle, only one argument is supported. When passing by value, multiple arguments are supported.

expose_function

Added in: v1.8

The method adds a function called `name` on the `window` object of every frame in the page. When called, the function executes `callback` and returns a Promise which resolves to the return value of `callback`.

If the `callback` returns a Promise, it will be awaited.

See browser_context.expose_function() for context-wide exposed function.

> **NOTE**
>
> Functions installed via page.expose_function() survive navigations.

Usage

An example of adding a `sha256` function to the page:

Sync Async

```
import hashlib
from playwright.sync_api import sync_playwright, Playwright

def sha256(text):
    m = hashlib.sha256()
    m.update(bytes(text, "utf8"))
    return m.hexdigest()


def run(playwright: Playwright):
    webkit = playwright.webkit
    browser = webkit.launch(headless=False)
    page = browser.new_page()
    page.expose_function("sha256", sha256)
    page.set_content("""
        <script>
          async function onClick() {
            document.querySelector('div').textContent = await
window.sha256('PLAYWRIGHT');
          }
        </script>
        <button onclick="onClick()">Click me</button>
        <div></div>
    """)
    page.click("button")

with sync_playwright() as playwright:
    run(playwright)
```

Arguments

- `name` str

  Name of the function on the window object

- `callback` Callable

  Callback function which will be called in Playwright's context.

frame

Added in: v1.8

Returns frame matching the specified criteria. Either `name` or `url` must be specified.

Usage

```
frame = page.frame(name="frame-name")
```

```
frame = page.frame(url=r".*domain.*")
```

Arguments

- `name` str *(optional)*

  Frame name specified in the `iframe`'s `name` attribute. Optional.

- `url` str|Pattern|Callable[URL]:bool *(optional)*

  A glob pattern, regex pattern or predicate receiving frame's `url` as a URL object. Optional.

Returns

- NoneType|Frame

frame_locator

Added in: v1.17

When working with iframes, you can create a frame locator that will enter the iframe and allow selecting elements in that iframe.

Usage

Following snippet locates element with text "Submit" in the iframe with id `my-frame`, like `<iframe id="my-frame">`:

Sync **Async**

```
locator = page.frame_locator("#my-iframe").get_by_text("Submit")
locator.click()
```

Arguments

- `selector` str

  A selector to use when resolving DOM element.

Returns

- FrameLocator

get_by_alt_text

Added in: v1.27

Allows locating elements by their alt text.

Usage

For example, this method will find the image by alt text "Playwright logo":

```
<img alt='Playwright logo'>
```

Sync Async

```
page.get_by_alt_text("Playwright logo").click()
```

Arguments

- `text` str|Pattern

  Text to locate the element for.

- `exact` bool *(optional)*

  Whether to find an exact match: case-sensitive and whole-string. Default to false. Ignored when locating by a regular expression. Note that exact match still trims whitespace.

Returns

- Locator

get_by_label

Added in: v1.27

Allows locating input elements by the text of the associated `<label>` or `aria-labelledby` element, or by the `aria-label` attribute.

Usage

For example, this method will find inputs by label "Username" and "Password" in the following DOM:

```
<input aria-label="Username">
<label for="password-input">Password:</label>
<input id="password-input">
```

Sync **Async**

```
page.get_by_label("Username").fill("john")
page.get_by_label("Password").fill("secret")
```

Arguments

- `text` str|Pattern

  Text to locate the element for.

- `exact` bool *(optional)*

  Whether to find an exact match: case-sensitive and whole-string. Default to false. Ignored when locating by a regular expression. Note that exact match still trims whitespace.

Returns

- Locator

get_by_placeholder

Added in: v1.27

Allows locating input elements by the placeholder text.

Usage

For example, consider the following DOM structure.

```
<input type="email" placeholder="name@example.com" />
```

You can fill the input after locating it by the placeholder text:

Sync | Async

```
page.get_by_placeholder("name@example.com").fill("playwright@microsoft.com")
```

Arguments

- `text` str|Pattern

  Text to locate the element for.

- `exact` bool *(optional)*

  Whether to find an exact match: case-sensitive and whole-string. Default to false. Ignored when locating by a regular expression. Note that exact match still trims whitespace.

Returns

- Locator

get_by_role

Added in: v1.27

Allows locating elements by their ARIA role, ARIA attributes and accessible name.

Usage

Consider the following DOM structure.

```
<h3>Sign up</h3>
<label>
  <input type="checkbox" /> Subscribe
</label>
<br/>
<button>Submit</button>
```

You can locate each element by it's implicit role:

Sync **Async**

```
expect(page.get_by_role("heading", name="Sign up")).to_be_visible()

page.get_by_role("checkbox", name="Subscribe").check()

page.get_by_role("button", name=re.compile("submit", re.IGNORECASE)).click()
```

Arguments

- `role` "alert"|"alertdialog"|"application"|"article"|"banner"|"blockquote"|"button"|"caption"|"cell"|"checkbox"|"code"|"columnheader"|"combobox"|"complementary"|"contentinfo"|"definition"|"deletion"|"dialog"|"directory"|"document"|"emphasis"|"feed"|"figure"|"form"|"generic"|"grid"|"gridcell"|"group"|"heading"|"img"|"insertion"|"link"|"list"|"listbox"|"listitem"|"log"|"main"|"marquee"|"math"|"meter"|"menu"|"menubar"|"menuitem"|"menuitemcheckbox"|"menuitemradio"|"navigation"|"none"|"note"|"option"|"paragraph"|"presentation"|"progressbar"|"radio"|"radiogroup"|"region"|"row"|"rowgroup"|"rowheader"|"scrollbar"|"search"|"searchbox"|"separator"|"slider"|"spinbutton"|"status"|"strong"|"subscript"|"superscript"|"switch"|"tab"|"table"|"tablist"|"tabpanel"|"term"|"textbox"|"time"|"timer"|"toolbar"|"tooltip"|"tree"|"treegrid"|"treeitem"

  Required aria role.

- `checked` bool *(optional)*

  An attribute that is usually set by `aria-checked` or native `<input type=checkbox>` controls.

  Learn more about `aria-checked`.

- `disabled` bool *(optional)*

An attribute that is usually set by `aria-disabled` or `disabled`.

> **NOTE**
>
> Unlike most other attributes, `disabled` is inherited through the DOM hierarchy. Learn more about `aria-disabled`.

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

  Whether `name` is matched exactly: case-sensitive and whole-string. Defaults to false. Ignored when `name` is a regular expression. Note that exact match still trims whitespace.

- `expanded` bool *(optional)*

  An attribute that is usually set by `aria-expanded`.

  Learn more about `aria-expanded`.

- `include_hidden` bool *(optional)*

  Option that controls whether hidden elements are matched. By default, only non-hidden elements, as defined by ARIA, are matched by role selector.

  Learn more about `aria-hidden`.

- `level` int *(optional)*

  A number attribute that is usually present for roles `heading`, `listitem`, `row`, `treeitem`, with default values for `<h1>-<h6>` elements.

  Learn more about `aria-level`.

- `name` str|Pattern *(optional)*

  Option to match the accessible name. By default, matching is case-insensitive and searches for a substring, use `exact` to control this behavior.

  Learn more about accessible name.

- `pressed` bool *(optional)*

  An attribute that is usually set by `aria-pressed`.

  Learn more about `aria-pressed`.

- `selected` bool *(optional)*

  An attribute that is usually set by `aria-selected`.

  Learn more about `aria-selected`.

Returns

- Locator

Details

Role selector **does not replace** accessibility audits and conformance tests, but rather gives early feedback about the ARIA guidelines.

Many html elements have an implicitly defined role that is recognized by the role selector. You can find all the supported roles here. ARIA guidelines **do not recommend** duplicating implicit roles and attributes by setting `role` and/or `aria-*` attributes to default values.

get_by_test_id

Added in: v1.27

Locate element by the test id.

Usage

Consider the following DOM structure.

```
<button data-testid="directions">Itinéraire</button>
```

You can locate the element by it's test id:

Sync | Async

```
page.get_by_test_id("directions").click()
```

Arguments

- `test_id` str|Pattern

  Id to locate the element by.

Returns

- Locator

Details

By default, the `data-testid` attribute is used as a test id. Use selectors.set_test_id_attribute() to configure a different test id attribute if necessary.

get_by_text

Added in: v1.27

Allows locating elements that contain given text.

See also locator.filter() that allows to match by another criteria, like an accessible role, and then filter by the text content.

Usage

Consider the following DOM structure:

```
<div>Hello <span>world</span></div>
<div>Hello</div>
```

You can locate by text substring, exact string, or a regular expression:

Sync **Async**

```
# Matches <span>
page.get_by_text("world")

# Matches first <div>
page.get_by_text("Hello world")

# Matches second <div>
```

```
page.get_by_text("Hello", exact=True)

# Matches both <div>s
page.get_by_text(re.compile("Hello"))

# Matches second <div>
page.get_by_text(re.compile("^hello$", re.IGNORECASE))
```

Arguments

- `text` str|Pattern

  Text to locate the element for.

- `exact` bool *(optional)*

  Whether to find an exact match: case-sensitive and whole-string. Default to false. Ignored when locating by a regular expression. Note that exact match still trims whitespace.

Returns

- Locator

Details

Matching by text always normalizes whitespace, even with exact match. For example, it turns multiple spaces into one, turns line breaks into spaces and ignores leading and trailing whitespace.

Input elements of the type `button` and `submit` are matched by their `value` instead of the text content. For example, locating by text `"Log in"` matches `<input type=button value="Log in">`.

get_by_title

Added in: v1.27

Allows locating elements by their title attribute.

Usage

Consider the following DOM structure.

```
<span title='Issues count'>25 issues</span>
```

You can check the issues count after locating it by the title text:

Sync Async

```
expect(page.get_by_title("Issues count")).to_have_text("25 issues")
```

Arguments

- `text` str|Pattern

  Text to locate the element for.

- `exact` bool *(optional)*

  Whether to find an exact match: case-sensitive and whole-string. Default to false. Ignored when locating by a regular expression. Note that exact match still trims whitespace.

Returns

- Locator

go_back

Added in: v1.8

Returns the main resource response. In case of multiple redirects, the navigation will resolve with the response of the last redirect. If can not go back, returns `null`.

Navigate to the previous page in history.

Usage

```
page.go_back()
page.go_back(**kwargs)
```

Arguments

- `timeout` float *(optional)*

  Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_navigation_timeout(), browser_context.set_default_timeout(), page.set_default_navigation_timeout() or page.set_default_timeout() methods.

- `wait_until` "load"|"domcontentloaded"|"networkidle"|"commit" *(optional)*

  When to consider operation succeeded, defaults to `load`. Events can be either:

  - `'domcontentloaded'` - consider operation to be finished when the `DOMContentLoaded` event is fired.
  - `'load'` - consider operation to be finished when the `load` event is fired.
  - `'networkidle'` - **DISCOURAGED** consider operation to be finished when there are no network connections for at least `500` ms. Don't use this method for testing, rely on web assertions to assess readiness instead.
  - `'commit'` - consider operation to be finished when network response is received and the document started loading.

Returns

- NoneType|Response

go_forward

Added in: v1.8

Returns the main resource response. In case of multiple redirects, the navigation will resolve with the response of the last redirect. If can not go forward, returns `null`.

Navigate to the next page in history.

Usage

```
page.go_forward()
page.go_forward(**kwargs)
```

Arguments

- `timeout` float *(optional)*

  Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_navigation_timeout(), browser_context.set_default_timeout(), page.set_default_navigation_timeout() or page.set_default_timeout() methods.

- `wait_until` "load"|"domcontentloaded"|"networkidle"|"commit" *(optional)*

  When to consider operation succeeded, defaults to `load`. Events can be either:

  - `'domcontentloaded'` - consider operation to be finished when the `DOMContentLoaded` event is fired.
  - `'load'` - consider operation to be finished when the `load` event is fired.
  - `'networkidle'` - **DISCOURAGED** consider operation to be finished when there are no network connections for at least `500` ms. Don't use this method for testing, rely on web assertions to assess readiness instead.
  - `'commit'` - consider operation to be finished when network response is received and the document started loading.

Returns

- NoneType|Response

goto

Added in: v1.8

Returns the main resource response. In case of multiple redirects, the navigation will resolve with the first non-redirect response.

The method will throw an error if:

- there's an SSL error (e.g. in case of self-signed certificates).
- target URL is invalid.
- the `timeout` is exceeded during navigation.
- the remote server does not respond or is unreachable.
- the main resource failed to load.

The method will not throw an error when any valid HTTP status code is returned by the remote server, including 404 "Not Found" and 500 "Internal Server Error". The status code for such responses can be retrieved by calling response.status.

> **NOTE**
>
> The method either throws an error or returns a main resource response. The only exceptions are navigation to `about:blank` or navigation to the same URL with a different hash, which would succeed and return `null`.

> **NOTE**
>
> Headless mode doesn't support navigation to a PDF document. See the upstream issue.

Usage

```
page.goto(url)
page.goto(url, **kwargs)
```

Arguments

- `url` str

  URL to navigate page to. The url should include scheme, e.g. `https://`. When a `base_url` via the context options was provided and the passed URL is a path, it gets merged via the `new URL()` constructor.

- `referer` str *(optional)*

  Referer header value. If provided it will take preference over the referer header value set by page.set_extra_http_headers().

- `timeout` float *(optional)*

  Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_navigation_timeout(), browser_context.set_default_timeout(), page.set_default_navigation_timeout() or page.set_default_timeout() methods.

- `wait_until` "load"|"domcontentloaded"|"networkidle"|"commit" *(optional)*

When to consider operation succeeded, defaults to `load`. Events can be either:

- `'domcontentloaded'` - consider operation to be finished when the `DOMContentLoaded` event is fired.
- `'load'` - consider operation to be finished when the `load` event is fired.
- `'networkidle'` - **DISCOURAGED** consider operation to be finished when there are no network connections for at least `500` ms. Don't use this method for testing, rely on web assertions to assess readiness instead.
- `'commit'` - consider operation to be finished when network response is received and the document started loading.

Returns

- NoneType|Response

locator

Added in: v1.14

The method returns an element locator that can be used to perform actions on this page / frame. Locator is resolved to the element immediately before performing an action, so a series of actions on the same locator can in fact be performed on different DOM elements. That would happen if the DOM structure between those actions has changed.

Learn more about locators.

Usage

```
page.locator(selector)
page.locator(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to use when resolving DOM element.

- `has` Locator *(optional)*

  Matches elements containing an element that matches an inner locator. Inner locator is queried against the outer one. For example, `article` that has `text=Playwright` matches

`<article><div>Playwright</div></article>`.

Note that outer and inner locators must belong to the same frame. Inner locator must not contain FrameLocators.

- `has_not` Locator *(optional)* Added in: v1.33

  Matches elements that do not contain an element that matches an inner locator. Inner locator is queried against the outer one. For example, `article` that does not have `div` matches `<article><span>Playwright</span></article>`.

  Note that outer and inner locators must belong to the same frame. Inner locator must not contain FrameLocators.

- `has_not_text` str|Pattern *(optional)* Added in: v1.33

  Matches elements that do not contain specified text somewhere inside, possibly in a child or a descendant element. When passed a [string], matching is case-insensitive and searches for a substring.

- `has_text` str|Pattern *(optional)*

  Matches elements containing specified text somewhere inside, possibly in a child or a descendant element. When passed a [string], matching is case-insensitive and searches for a substring. For example, `"Playwright"` matches `<article><div>Playwright</div></article>`.

Returns

- Locator

opener

Added in: v1.8

Returns the opener for popup pages and `null` for others. If the opener has been closed already the returns `null`.

Usage

```
page.opener()
```

Returns

- NoneType|Page

pause

Added in: v1.9

Pauses script execution. Playwright will stop executing the script and wait for the user to either press 'Resume' button in the page overlay or to call `playwright.resume()` in the DevTools console.

User can inspect selectors or perform manual steps while paused. Resume will continue running the original script from the place it was paused.

> **NOTE**
>
> This method requires Playwright to be started in a headed mode, with a falsy `headless` value in the browser_type.launch().

Usage

```
page.pause()
```

pdf

Added in: v1.8

Returns the PDF buffer.

> **NOTE**
>
> Generating a pdf is currently only supported in Chromium headless.

`page.pdf()` generates a pdf of the page with `print` css media. To generate a pdf with `screen` media, call page.emulate_media() before calling `page.pdf()`:

> **NOTE**

> By default, `page.pdf()` generates a pdf with modified colors for printing. Use the `-webkit-print-color-adjust` property to force rendering of exact colors.

Usage

Sync **Async**

```
# generates a pdf with "screen" media type.
page.emulate_media(media="screen")
page.pdf(path="page.pdf")
```

The `width`, `height`, and `margin` options accept values labeled with units. Unlabeled values are treated as pixels.

A few examples:

- `page.pdf({width: 100})` - prints with width set to 100 pixels
- `page.pdf({width: '100px'})` - prints with width set to 100 pixels
- `page.pdf({width: '10cm'})` - prints with width set to 10 centimeters.

All possible units are:

- `px` - pixel
- `in` - inch
- `cm` - centimeter
- `mm` - millimeter

The `format` options are:

- `Letter`: 8.5in x 11in
- `Legal`: 8.5in x 14in
- `Tabloid`: 11in x 17in
- `Ledger`: 17in x 11in
- `A0`: 33.1in x 46.8in
- `A1`: 23.4in x 33.1in
- `A2`: 16.54in x 23.4in
- `A3`: 11.7in x 16.54in
- `A4`: 8.27in x 11.7in

- `A5`: 5.83in x 8.27in
- `A6`: 4.13in x 5.83in

> **NOTE**
> `header_template` and `footer_template` markup have the following limitations: > 1. Script tags inside templates are not evaluated. > 2. Page styles are not visible inside templates.

Arguments

- `display_header_footer` bool *(optional)*

  Display header and footer. Defaults to `false`.

- `footer_template` str *(optional)*

  HTML template for the print footer. Should use the same format as the `header_template`.

- `format` str *(optional)*

  Paper format. If set, takes priority over `width` or `height` options. Defaults to 'Letter'.

- `header_template` str *(optional)*

  HTML template for the print header. Should be valid HTML markup with following classes used to inject printing values into them:

  - `'date'` formatted print date
  - `'title'` document title
  - `'url'` document location
  - `'pageNumber'` current page number
  - `'totalPages'` total pages in the document

- `height` str|float *(optional)*

  Paper height, accepts values labeled with units.

- `landscape` bool *(optional)*

  Paper orientation. Defaults to `false`.

- `margin` Dict *(optional)*

- `top` str|float *(optional)*

  Top margin, accepts values labeled with units. Defaults to `0`.

- `right` str|float *(optional)*

  Right margin, accepts values labeled with units. Defaults to `0`.

- `bottom` str|float *(optional)*

  Bottom margin, accepts values labeled with units. Defaults to `0`.

- `left` str|float *(optional)*

  Left margin, accepts values labeled with units. Defaults to `0`.

Paper margins, defaults to none.

- `page_ranges` str *(optional)*

  Paper ranges to print, e.g., '1-5, 8, 11-13'. Defaults to the empty string, which means print all pages.

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

  The file path to save the PDF to. If `path` is a relative path, then it is resolved relative to the current working directory. If no path is provided, the PDF won't be saved to the disk.

- `prefer_css_page_size` bool *(optional)*

  Give any CSS `@page` size declared in the page priority over what is declared in `width` and `height` or `format` options. Defaults to `false`, which will scale the content to fit the paper size.

- `print_background` bool *(optional)*

  Print background graphics. Defaults to `false`.

- `scale` float *(optional)*

  Scale of the webpage rendering. Defaults to `1`. Scale amount must be between 0.1 and 2.

- `width` str|float *(optional)*

Paper width, accepts values labeled with units.

Returns

- bytes

reload

Added in: v1.8

This method reloads the current page, in the same way as if the user had triggered a browser refresh. Returns the main resource response. In case of multiple redirects, the navigation will resolve with the response of the last redirect.

Usage

```
page.reload()
page.reload(**kwargs)
```

Arguments

- `timeout` float *(optional)*

  Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_navigation_timeout(), browser_context.set_default_timeout(), page.set_default_navigation_timeout() or page.set_default_timeout() methods.

- `wait_until` "load"|"domcontentloaded"|"networkidle"|"commit" *(optional)*

  When to consider operation succeeded, defaults to `load`. Events can be either:

  - `'domcontentloaded'` - consider operation to be finished when the `DOMContentLoaded` event is fired.
  - `'load'` - consider operation to be finished when the `load` event is fired.
  - `'networkidle'` - **DISCOURAGED** consider operation to be finished when there are no network connections for at least `500` ms. Don't use this method for testing, rely on web assertions to assess readiness instead.
  - `'commit'` - consider operation to be finished when network response is received and the document started loading.

Returns

- NoneType|Response

route

Added in: v1.8

Routing provides the capability to modify network requests that are made by a page.

Once routing is enabled, every request matching the url pattern will stall unless it's continued, fulfilled or aborted.

> **NOTE**
>
> The handler will only be called for the first url if the response is a redirect.

> **NOTE**
>
> page.route() will not intercept requests intercepted by Service Worker. See this issue. We recommend disabling Service Workers when using request interception by setting `browser.new_context.service_workers` to `'block'`.

Usage

An example of a naive handler that aborts all image requests:

Sync **Async**

```
page = browser.new_page()
page.route("**/*.{png,jpg,jpeg}", lambda route: route.abort())
page.goto("https://example.com")
browser.close()
```

or the same snippet using a regex pattern instead:

Sync **Async**

```
page = browser.new_page()
page.route(re.compile(r"(\.png$)|(\.jpg$)"), lambda route: route.abort())
page.goto("https://example.com")
browser.close()
```

It is possible to examine the request to decide the route action. For example, mocking all requests that contain some post data, and leaving all other requests as is:

Sync **Async**

```
def handle_route(route):
  if ("my-string" in route.request.post_data):
    route.fulfill(body="mocked-data")
  else:
    route.continue_()
page.route("/api/**", handle_route)
```

Page routes take precedence over browser context routes (set up with browser_context.route()) when request matches both handlers.

To remove a route with its handler you can use page.unroute().

> **NOTE**
>
> Enabling routing disables http cache.

Arguments

- `url` str|Pattern|Callable[URL]:bool

  A glob pattern, regex pattern or predicate receiving URL to match while routing. When a `base_url` via the context options was provided and the passed URL is a path, it gets merged via the `new URL()` constructor.

- `handler` Callable[Route, Request]:Promise[Any]|Any

  handler function to route the request.

- `times` int *(optional)* Added in: v1.15

How often a route should be used. By default it will be used every time.

route_from_har

Added in: v1.23

If specified the network requests that are made in the page will be served from the HAR file. Read more about Replaying from HAR.

Playwright will not serve requests intercepted by Service Worker from the HAR file. See this issue. We recommend disabling Service Workers when using request interception by setting `browser.new_context.service_workers` to `'block'`.

Usage

```
page.route_from_har(har)
page.route_from_har(har, **kwargs)
```

Arguments

- `har` Union[str, pathlib.Path]

  Path to a HAR file with prerecorded network data. If `path` is a relative path, then it is resolved relative to the current working directory.

- `not_found` "abort"|"fallback" *(optional)*

  - If set to 'abort' any request not found in the HAR file will be aborted.
  - If set to 'fallback' missing requests will be sent to the network.

  Defaults to abort.

- `update` bool *(optional)*

  If specified, updates the given HAR with the actual network information instead of serving from file. The file is written to disk when browser_context.close() is called.

- `update_content` "embed"|"attach" *(optional)* Added in: v1.32

  Optional setting to control resource content management. If `attach` is specified, resources are persisted as separate files or entries in the ZIP archive. If `embed` is specified, content is stored inline the HAR file.

- `update_mode` "full"|"minimal" *(optional)* Added in: v1.32

  When set to `minimal`, only record information necessary for routing from HAR. This omits sizes, timing, page, cookies, security and other types of HAR information that are not used when replaying from HAR. Defaults to `full`.

- `url` str|Pattern *(optional)*

  A glob pattern, regular expression or predicate to match the request URL. Only requests with URL matching the pattern will be served from the HAR file. If not specified, all requests are served from the HAR file.

screenshot

Added in: v1.8

Returns the buffer with the captured screenshot.

Usage

```
page.screenshot()
page.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"`.

- `clip` Dict *(optional)*
  - `x` float

    x-coordinate of top-left corner of clip area
  - `y` float

    y-coordinate of top-left corner of clip area
  - `width` float

    width of clipping area
  - `height` float

    height of clipping area

  An object which specifies clipping of the resulting image.
- `full_page` bool *(optional)*

  When true, takes a screenshot of the full scrollable page, instead of the currently visible viewport. Defaults to `false`.
- `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

set_content

Added in: v1.8

This method internally calls document.write(), inheriting all its specific characteristics and behaviors.

Usage

```
page.set_content(html)
page.set_content(html, **kwargs)
```

Arguments

- `html` str

  HTML markup to assign to the page.

- `timeout` float *(optional)*

  Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_navigation_timeout(), browser_context.set_default_timeout(), page.set_default_navigation_timeout() or page.set_default_timeout() methods.

- `wait_until` "load"|"domcontentloaded"|"networkidle"|"commit" *(optional)*

  When to consider operation succeeded, defaults to `load`. Events can be either:

  - `'domcontentloaded'` - consider operation to be finished when the `DOMContentLoaded` event is fired.
  - `'load'` - consider operation to be finished when the `load` event is fired.
  - `'networkidle'` - **DISCOURAGED** consider operation to be finished when there are no network connections for at least `500` ms. Don't use this method for testing, rely on web assertions to assess readiness instead.
  - `'commit'` - consider operation to be finished when network response is received and the document started loading.

set_default_navigation_timeout

Added in: v1.8

This setting will change the default maximum navigation time for the following methods and related shortcuts:

- page.go_back()
- page.go_forward()
- page.goto()
- page.reload()
- page.set_content()
- page.expect_navigation()
- page.wait_for_url()

> **NOTE**
>
> page.set_default_navigation_timeout() takes priority over page.set_default_timeout(), browser_context.set_default_timeout() and browser_context.set_default_navigation_timeout().

Usage

```
page.set_default_navigation_timeout(timeout)
```

Arguments

- `timeout` float

  Maximum navigation time in milliseconds

set_default_timeout

Added in: v1.8

This setting will change the default maximum time for all the methods accepting `timeout` option.

> **NOTE**
>
> page.set_default_navigation_timeout() takes priority over page.set_default_timeout().

Usage

```
page.set_default_timeout(timeout)
```

Arguments

- `timeout` float

  Maximum time in milliseconds

set_extra_http_headers

Added in: v1.8

The extra HTTP headers will be sent with every request the page initiates.

> **NOTE**
>
> page.set_extra_http_headers() does not guarantee the order of headers in the outgoing requests.

Usage

```
page.set_extra_http_headers(headers)
```

Arguments

- `headers` Dict[str, str]

  An object containing additional HTTP headers to be sent with every request. All header values must be strings.

set_viewport_size

Added in: v1.8

In the case of multiple pages in a single browser, each page can have its own viewport size. However, browser.new_context() allows to set viewport size (and more) for all pages in the context at once.

page.set_viewport_size() will resize the page. A lot of websites don't expect phones to change size, so you should set the viewport size before navigating to the page. page.set_viewport_size() will also reset `screen` size, use browser.new_context() with `screen` and `viewport` parameters if you need better control of these properties.

Usage

Sync **Async**

```
page = browser.new_page()
page.set_viewport_size({"width": 640, "height": 480})
page.goto("https://example.com")
```

Arguments

- `viewport_size` Dict
  - `width` int

    page width in pixels.
  - `height` int

    page height in pixels.

title

Added in: v1.8

Returns the page's title.

Usage

```
page.title()
```

Returns

- str

unroute

Added in: v1.8

Removes a route created with page.route(). When `handler` is not specified, removes all routes for the `url`.

Usage

```
page.unroute(url)
page.unroute(url, **kwargs)
```

Arguments

- `url` str|Pattern|Callable[URL]:bool

A glob pattern, regex pattern or predicate receiving URL to match while routing.

- `handler` Callable[Route, Request]:Promise[Any]|Any *(optional)*

  Optional handler function to route the request.

wait_for_event

Added in: v1.8

> **NOTE**
>
> In most cases, you should use page.expect_event().

Waits for given `event` to fire. If predicate is provided, it passes event's value into the `predicate` function and waits for `predicate(event)` to return a truthy value. Will throw an error if the page is closed before the `event` is fired.

Usage

```
page.wait_for_event(event)
page.wait_for_event(event, **kwargs)
```

Arguments

- `event` str

  Event name, same one typically passed into `*.on(event)`.

- `predicate` Callable *(optional)*

  Receives the event data and resolves to truthy value when the waiting should resolve.

- `timeout` float *(optional)*

  Maximum time to wait for 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().

Returns

- Any

wait_for_function

Added in: v1.8

Returns when the `expression` returns a truthy value. It resolves to a JSHandle of the truthy value.

Usage

The page.wait_for_function() can be used to observe viewport size change:

Sync **Async**

```
from playwright.sync_api import sync_playwright, Playwright

def run(playwright: Playwright):
    webkit = playwright.webkit
    browser = webkit.launch()
    page = browser.new_page()
    page.evaluate("window.x = 0; setTimeout(() => { window.x = 100 }, 1000);")
    page.wait_for_function("() => window.x > 0")
    browser.close()

with sync_playwright() as playwright:
    run(playwright)
```

To pass an argument to the predicate of page.wait_for_function() function:

Sync **Async**

```
selector = ".foo"
page.wait_for_function("selector => !!document.querySelector(selector)", selector)
```

Arguments

- `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`.

- `polling` float|"raf" *(optional)*

  If `polling` is `'raf'`, then `expression` is constantly executed in `requestAnimationFrame` callback. If `polling` is a number, then it is treated as an interval in milliseconds at which the function would be executed. Defaults to `raf`.

- `timeout` float *(optional)*

  Maximum time to wait for 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

- JSHandle

wait_for_load_state

Added in: v1.8

Returns when the required load state has been reached.

This resolves when the page reaches a required load state, `load` by default. The navigation must have been committed when this method is called. If current document has already reached the required state, resolves immediately.

Usage

Sync Async

```
page.get_by_role("button").click() # click triggers navigation.
page.wait_for_load_state() # the promise resolves after "load" event.
```

Sync Async

```python
with page.expect_popup() as page_info:
    page.get_by_role("button").click() # click triggers a popup.
popup = page_info.value
# Wait for the "DOMContentLoaded" event.
popup.wait_for_load_state("domcontentloaded")
print(popup.title()) # popup is ready to use.
```

Arguments

- `state` "load"|"domcontentloaded"|"networkidle" *(optional)*

  Optional load state to wait for, defaults to `load`. If the state has been already reached while loading current document, the method resolves immediately. Can be one of:

  - `'load'` - wait for the `load` event to be fired.
  - `'domcontentloaded'` - wait for the `DOMContentLoaded` event to be fired.
  - `'networkidle'` - **DISCOURAGED** wait until there are no network connections for at least `500` ms. Don't use this method for testing, rely on web assertions to assess readiness instead.

- `timeout` float *(optional)*

  Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_navigation_timeout(), browser_context.set_default_timeout(), page.set_default_navigation_timeout() or page.set_default_timeout() methods.

wait_for_url

Added in: v1.11

Waits for the main frame to navigate to the given URL.

Usage

Sync **Async**

```python
page.click("a.delayed-navigation") # clicking the link will indirectly cause a navigation
```

```
page.wait_for_url("**/target.html")
```

Arguments

- `url` str|Pattern|Callable[URL]:bool

  A glob pattern, regex pattern or predicate receiving URL to match while waiting for the navigation. Note that if the parameter is a string without wildcard characters, the method will wait for navigation to URL that is exactly equal to the string.

- `timeout` float *(optional)*

  Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_navigation_timeout(), browser_context.set_default_timeout(), page.set_default_navigation_timeout() or page.set_default_timeout() methods.

- `wait_until` "load"|"domcontentloaded"|"networkidle"|"commit" *(optional)*

  When to consider operation succeeded, defaults to `load`. Events can be either:

  - `'domcontentloaded'` - consider operation to be finished when the `DOMContentLoaded` event is fired.
  - `'load'` - consider operation to be finished when the `load` event is fired.
  - `'networkidle'` - **DISCOURAGED** consider operation to be finished when there are no network connections for at least `500` ms. Don't use this method for testing, rely on web assertions to assess readiness instead.
  - `'commit'` - consider operation to be finished when network response is received and the document started loading.

Properties

context

Added in: v1.8

Get the browser context that the page belongs to.

Usage

```
page.context
```

Returns

- BrowserContext

frames

Added in: v1.8

An array of all frames attached to the page.

Usage

```
page.frames
```

Returns

- List[Frame]

is_closed

Added in: v1.8

Indicates that the page has been closed.

Usage

```
page.is_closed()
```

Returns

- bool

keyboard

Added in: v1.8

Usage

```
page.keyboard
```

Type

- Keyboard

main_frame

Added in: v1.8

The page's main frame. Page is guaranteed to have a main frame which persists during navigations.

Usage

```
page.main_frame
```

Returns

- Frame

mouse

Added in: v1.8

Usage

```
page.mouse
```

Type

- Mouse

request

Added in: v1.16

API testing helper associated with this page. This method returns the same instance as browser_context.request on the page's context. See browser_context.request for more details.

Usage

```
page.request
```

Type

- APIRequestContext

touchscreen

Added in: v1.8

Usage

```
page.touchscreen
```

Type

- Touchscreen

url

Added in: v1.8

Usage

```
page.url
```

Returns

- str

video

Added in: v1.8

Video object associated with this page.

Usage

```
page.video
```

Returns

- NoneType|Video

viewport_size

Added in: v1.8

Usage

```
page.viewport_size
```

Returns

- NoneType|Dict
  - `width` int

    page width in pixels.
  - `height` int

    page height in pixels.

workers

Added in: v1.8

This method returns all of the dedicated WebWorkers associated with the page.

> **NOTE**
>
> This does not contain ServiceWorkers

Usage

```
page.workers
```

Returns

- List[Worker]

Events

on("close")

Added in: v1.8

Emitted when the page closes.

Usage

```
page.on("close", handler)
```

Event data

- Page

on("console")

Added in: v1.8

Emitted when JavaScript within the page calls one of console API methods, e.g. `console.log` or `console.dir`. Also emitted if the page throws an error or a warning.

The arguments passed into `console.log` are available on the ConsoleMessage event handler argument.

Usage

Sync | Async

```
def print_args(msg):
    for arg in msg.args:
        print(arg.json_value())
```

```
page.on("console", print_args)
page.evaluate("console.log('hello', 5, { foo: 'bar' })")
```

Event data

- ConsoleMessage

on("crash")

Added in: v1.8

Emitted when the page crashes. Browser pages might crash if they try to allocate too much memory. When the page crashes, ongoing and subsequent operations will throw.

The most common way to deal with crashes is to catch an exception:

Sync | Async

```
try:
    # crash might happen during a click.
    page.click("button")
    # or while waiting for an event.
    page.wait_for_event("popup")
except Error as e:
    pass
    # when the page crashes, exception message contains "crash".
```

Usage

```
page.on("crash", handler)
```

Event data

- Page

on("dialog")

Added in: v1.8

Emitted when a JavaScript dialog appears, such as `alert`, `prompt`, `confirm` or `beforeunload`. Listener **must** either dialog.accept() or dialog.dismiss() the dialog - otherwise the page will freeze waiting for the dialog, and actions like click will never finish.

Usage

```
page.on("dialog", lambda dialog: dialog.accept())
```

> **NOTE**
>
> When no page.on("dialog") or browser_context.on("dialog") listeners are present, all dialogs are automatically dismissed.

Event data

- Dialog

on("domcontentloaded")

Added in: v1.9

Emitted when the JavaScript `DOMContentLoaded` event is dispatched.

Usage

```
page.on("domcontentloaded", handler)
```

Event data

- Page

on("download")

Added in: v1.8

Emitted when attachment download started. User can access basic file operations on downloaded content via the passed Download instance.

Usage

```
page.on("download", handler)
```

Event data

- Download

on("filechooser")

Added in: v1.9

Emitted when a file chooser is supposed to appear, such as after clicking the `<input type=file>`. Playwright can respond to it via setting the input files using file_chooser.set_files() that can be uploaded after that.

```
page.on("filechooser", lambda file_chooser:
file_chooser.set_files("/tmp/myfile.pdf"))
```

Usage

```
page.on("filechooser", handler)
```

Event data

- FileChooser

on("frameattached")

Added in: v1.9

Emitted when a frame is attached.

Usage

```
page.on("frameattached", handler)
```

Event data

- Frame

on("framedetached")

Added in: v1.9

Emitted when a frame is detached.

Usage

```
page.on("framedetached", handler)
```

Event data

- Frame

on("framenavigated")

Added in: v1.9

Emitted when a frame is navigated to a new url.

Usage

```
page.on("framenavigated", handler)
```

Event data

- Frame

on("load")

Added in: v1.8

Emitted when the JavaScript `load` event is dispatched.

Usage

```
page.on("load", handler)
```

Event data

- Page

on("pageerror")

Added in: v1.9

Emitted when an uncaught exception happens within the page.

Sync Async

```
# Log all uncaught errors to the terminal
page.on("pageerror", lambda exc: print(f"uncaught exception: {exc}"))

# Navigate to a page with an exception.
page.goto("data:text/html,<script>throw new Error('test')</script>")
```

Usage

```
page.on("pageerror", handler)
```

Event data

- Error

on("popup")

Added in: v1.8

Emitted when the page opens a new tab or window. This event is emitted in addition to the browser_context.on("page"), but only for popups relevant to this page.

The earliest moment that page is available is when it has navigated to the initial url. For example, when opening a popup with `window.open('http://example.com')`, this event will fire when the network request to "http://example.com" is done and its response has started loading in the popup.

Sync Async

```
with page.expect_event("popup") as page_info:
    page.get_by_text("open the popup").click()
popup = page_info.value
print(popup.evaluate("location.href"))
```

> **NOTE**
>
> Use page.wait_for_load_state() to wait until the page gets to a particular state (you should not need it in most cases).

Usage

```
page.on("popup", handler)
```

Event data

- Page

on("request")

Added in: v1.8

Emitted when a page issues a request. The request object is read-only. In order to intercept and mutate requests, see page.route() or browser_context.route().

Usage

```
page.on("request", handler)
```

Event data

- Request

on("requestfailed")

Added in: v1.9

Emitted when a request fails, for example by timing out.

```
page.on("requestfailed", lambda request: print(request.url + " " +
request.failure.error_text))
```

> **NOTE**
>
> HTTP Error responses, such as 404 or 503, are still successful responses from HTTP standpoint, so request will complete with page.on("requestfinished") event and not with page.on("requestfailed"). A request will only be considered failed when the client cannot get an HTTP response from the server, e.g. due to network error net::ERR_FAILED.

Usage

```
page.on("requestfailed", handler)
```

Event data

- Request

on("requestfinished")

Added in: v1.9

Emitted when a request finishes successfully after downloading the response body. For a successful response, the sequence of events is `request`, `response` and `requestfinished`.

Usage

```
page.on("requestfinished", handler)
```

Event data

- Request

on("response")

Added in: v1.8

Emitted when response status and headers are received for a request. For a successful response, the sequence of events is `request`, `response` and `requestfinished`.

Usage

```
page.on("response", handler)
```

Event data

- Response

on("websocket")

Added in: v1.9

Emitted when WebSocket request is sent.

Usage

```
page.on("websocket", handler)
```

Event data

- WebSocket

on("worker")

Added in: v1.8

Emitted when a dedicated WebWorker is spawned by the page.

Usage

```
page.on("worker", handler)
```

Event data

- Worker

Deprecated

accessibility

Added in: v1.8

> **DEPRECATED**
>
> This property is discouraged. Please use other libraries such as Axe if you need to test page accessibility. See our Node.js guide for integration with Axe.

Usage

```
page.accessibility
```

Type

- Accessibility

check

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.check() instead. Read more about locators.

This method checks an element matching `selector` by performing the following steps:

1. Find an element matching `selector`. If there is none, wait until a matching element is attached to the DOM.
2. Ensure that matched element is a checkbox or a radio input. If not, this method throws. If the element is already checked, 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. 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

```
page.check(selector)
page.check(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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.

- `strict` bool *(optional)* Added in: v1.14

  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.

- `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

> **DISCOURAGED**
>
> Use locator-based locator.click() instead. Read more about locators.

This method clicks an element matching `selector` by performing the following steps:

1. Find an element matching `selector`. If there is none, wait until a matching element is attached to the DOM.
2. 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.
3. Scroll the element into view if needed.
4. Use page.mouse to click in the center of the element, or the specified `position`.
5. Wait for initiated navigations to either succeed or fail, unless `no_wait_after` option is set.

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

Usage

```
page.click(selector)
page.click(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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.

- `strict` bool *(optional)* Added in: v1.14

  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.

- `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.

dblclick

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.dblclick() instead. Read more about locators.

This method double clicks an element matching `selector` by performing the following steps:

1. Find an element matching `selector`. If there is none, wait until a matching element is attached to the DOM.
2. 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.
3. Scroll the element into view if needed.
4. Use page.mouse to double click in the center of the element, or the specified `position`.
5. 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.

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

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

Usage

```
page.dblclick(selector)
page.dblclick(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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.

- `strict` bool *(optional)* Added in: v1.14

  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.

- `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

> **DISCOURAGED**
>
> Use locator-based locator.dispatch_event() instead. Read more about locators.

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

```
page.dispatch_event("button#submit", "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()")
page.dispatch_event("#source", "dragstart", { "dataTransfer": data_transfer })
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `type` str

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

- `event_init` EvaluationArgument *(optional)*

  Optional event-specific initialization properties.

- `strict` bool *(optional)* Added in: v1.14

  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.

eval_on_selector

Added in: v1.9

DISCOURAGED

This method does not wait for the element to pass actionability checks and therefore can lead to the flaky tests. Use locator.evaluate(), other Locator helper methods or web-first assertions instead.

The method finds an element matching the specified selector within the page and passes it as a first argument to `expression`. If no elements match the selector, the method throws an error. Returns the value of `expression`.

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

Usage

Sync **Async**

```
search_value = page.eval_on_selector("#search", "el => el.value")
preload_href = page.eval_on_selector("link[rel=preload]", "el => el.href")
html = page.eval_on_selector(".main-container", "(e, suffix) => e.outer_html +
suffix", "hello")
```

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`.

- `strict` bool *(optional)* Added in: v1.14

  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.

Returns

- Serializable

eval_on_selector_all

Added in: v1.9

> **DISCOURAGED**
>
> In most cases, locator.evaluate_all(), other Locator helper methods and web-first assertions do a better job.

The method finds all elements matching the specified selector within the page and passes an array of matched elements as a first argument to `expression`. Returns the result of `expression` invocation.

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

Usage

Sync | Async

```
div_counts = page.eval_on_selector_all("div", "(divs, min) => divs.length >= min",
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

expect_navigation

Added in: v1.8

> **DEPRECATED**
>
> This method is inherently racy, please use page.wait_for_url() instead.

Waits for the main frame navigation and returns the main resource response. In case of multiple redirects, the navigation will resolve with the response of the last redirect. In case of navigation to a different anchor or navigation due to History API usage, the navigation will resolve with `null`.

Usage

This resolves when the page navigates to a new URL or reloads. It is useful for when you run code which will indirectly cause the page to navigate. e.g. The click target has an `onclick` handler that triggers navigation from a `setTimeout`. Consider this example:

Sync **Async**

```
with page.expect_navigation():
    # This action triggers the navigation after a timeout.
    page.get_by_text("Navigate after timeout").click()
# Resolves after navigation has finished
```

> **NOTE**
>
> Usage of the History API to change the URL is considered a navigation.

Arguments

- `timeout` float *(optional)*

Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browser_context.set_default_navigation_timeout(), browser_context.set_default_timeout(), page.set_default_navigation_timeout() or page.set_default_timeout() methods.

- `url` str|Pattern|Callable[URL]:bool *(optional)*

  A glob pattern, regex pattern or predicate receiving URL to match while waiting for the navigation. Note that if the parameter is a string without wildcard characters, the method will wait for navigation to URL that is exactly equal to the string.

- `wait_until` "load"|"domcontentloaded"|"networkidle"|"commit" *(optional)*

  When to consider operation succeeded, defaults to `load`. Events can be either:

  - `'domcontentloaded'` - consider operation to be finished when the `DOMContentLoaded` event is fired.
  - `'load'` - consider operation to be finished when the `load` event is fired.
  - `'networkidle'` - **DISCOURAGED** consider operation to be finished when there are no network connections for at least `500` ms. Don't use this method for testing, rely on web assertions to assess readiness instead.
  - `'commit'` - consider operation to be finished when network response is received and the document started loading.

Returns

- EventContextManager[Response]

fill

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.fill() instead. Read more about locators.

This method waits for an element matching `selector`, 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

```
page.fill(selector, value)
page.fill(selector, value, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `value` str

  Value to fill 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`.

- `strict` bool *(optional)* Added in: v1.14

  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.

focus

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.focus() instead. Read more about locators.

This method fetches an element with `selector` and focuses it. If there's no element matching `selector`, the method waits until a matching element appears in the DOM.

Usage

```
page.focus(selector)
page.focus(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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.

get_attribute

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.get_attribute() instead. Read more about locators.

Returns element attribute value.

Usage

```
page.get_attribute(selector, name)
page.get_attribute(selector, name, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `name` str

  Attribute name to get the value for.

- `strict` bool *(optional)* Added in: v1.14

  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|str

hover

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.hover() instead. Read more about locators.

This method hovers over an element matching `selector` by performing the following steps:

1. Find an element matching `selector`. If there is none, wait until a matching element is attached to the DOM.
2. 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.
3. Scroll the element into view if needed.
4. Use page.mouse to hover over the center of the element, or the specified `position`.
5. Wait for initiated navigations to either succeed or fail, unless `noWaitAfter` option is set.

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

Usage

```
page.hover(selector)
page.hover(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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.

- `strict` bool *(optional)* Added in: v1.14

  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.

- `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

> **DISCOURAGED**
>
> Use locator-based locator.inner_html() instead. Read more about locators.

Returns `element.innerHTML`.

Usage

```
page.inner_html(selector)
page.inner_html(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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

- str

inner_text

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.inner_text() instead. Read more about locators.

Returns `element.innerText`.

Usage

```
page.inner_text(selector)
page.inner_text(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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

- str

input_value

Added in: v1.13

> **DISCOURAGED**
>
> Use locator-based locator.input_value() instead. Read more about locators.

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

```
page.input_value(selector)
page.input_value(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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

- str

is_checked

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.is_checked() instead. Read more about locators.

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

Usage

```
page.is_checked(selector)
page.is_checked(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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

- bool

is_disabled

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.is_disabled() instead. Read more about locators.

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

Usage

```
page.is_disabled(selector)
page.is_disabled(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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

- bool

is_editable

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.is_editable() instead. Read more about locators.

Returns whether the element is editable.

Usage

```
page.is_editable(selector)
page.is_editable(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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

- bool

is_enabled

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.is_enabled() instead. Read more about locators.

Returns whether the element is enabled.

Usage

```
page.is_enabled(selector)
page.is_enabled(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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

- bool

is_hidden

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.is_hidden() instead. Read more about locators.

Returns whether the element is hidden, the opposite of visible. `selector` that does not match any elements is considered hidden.

Usage

```
page.is_hidden(selector)
page.is_hidden(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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)*

  > **DEPRECATED**
  >
  > This option is ignored. page.is_hidden() does not wait for the element to become hidden and returns immediately.

Returns

- bool

is_visible

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.is_visible() instead. Read more about locators.

Returns whether the element is visible. `selector` that does not match any elements is considered not visible.

Usage

```
page.is_visible(selector)
```

```
page.is_visible(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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)*

  > **DEPRECATED**
  >
  > This option is ignored. page.is_visible() does not wait for the element to become visible and returns immediately.

Returns

- bool

press

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.press() instead. Read more about locators.

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

Sync **Async**

```
page = browser.new_page()
page.goto("https://keycode.info")
page.press("body", "A")
page.screenshot(path="a.png")
page.press("body", "ArrowLeft")
page.screenshot(path="arrow_left.png")
page.press("body", "Shift+O")
page.screenshot(path="o.png")
browser.close()
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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`.

- `strict` bool *(optional)* Added in: v1.14

  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.

query_selector

Added in: v1.9

> **DISCOURAGED**
>
> Use locator-based page.locator() instead. Read more about locators.

The method finds an element matching the specified selector within the page. If no elements match the selector, the return value resolves to `null`. To wait for an element on the page, use locator.wait_for().

Usage

```
page.query_selector(selector)
page.query_selector(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to query for.

- `strict` bool *(optional)* Added in: v1.14

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.

Returns

- NoneType|ElementHandle

query_selector_all

Added in: v1.9

> **DISCOURAGED**
>
> Use locator-based page.locator() instead. Read more about locators.

The method finds all elements matching the specified selector within the page. If no elements match the selector, the return value resolves to `[]`.

Usage

```
page.query_selector_all(selector)
```

Arguments

- `selector` str

  A selector to query for.

Returns

- List[ElementHandle]

select_option

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.select_option() instead. Read more about locators.

This method waits for an element matching `selector`, 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
page.select_option("select#colors", "blue")
# single selection matching both the label
page.select_option("select#colors", label="blue")
# multiple selection
page.select_option("select#colors", value=["red", "green", "blue"])
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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`.

- `strict` bool *(optional)* Added in: v1.14

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.

- `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]

set_checked

Added in: v1.15

> **DISCOURAGED**
>
> Use locator-based locator.set_checked() instead. Read more about locators.

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

1. Find an element matching `selector`. If there is none, wait until a matching element is attached to the DOM.
2. Ensure that matched element is a checkbox or a radio input. If not, this method throws.
3. If the element already has the right checked state, this method returns immediately.
4. 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.
5. Scroll the element into view if needed.
6. Use page.mouse to click in the center of the element.
7. Wait for initiated navigations to either succeed or fail, unless `no_wait_after` option is set.
8. 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

```
page.set_checked(selector, checked)
page.set_checked(selector, checked, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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.

- `strict` bool *(optional)*

  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.

- `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

> **DISCOURAGED**
>
> Use locator-based locator.set_input_files() instead. Read more about locators.

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 `selector` 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

```
page.set_input_files(selector, files)
page.set_input_files(selector, files, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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`.

- `strict` bool *(optional)* Added in: v1.14

  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.

tap

Added in: v1.8

> ⚠ **DISCOURAGED**

Use locator-based locator.tap() instead. Read more about locators.

This method taps an element matching `selector` by performing the following steps:

1. Find an element matching `selector`. If there is none, wait until a matching element is attached to the DOM.
2. 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.
3. Scroll the element into view if needed.
4. Use page.touchscreen to tap the center of the element, or the specified `position`.
5. Wait for initiated navigations to either succeed or fail, unless `no_wait_after` option is set.

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

> **NOTE**
>
> page.tap() the method will throw if `has_touch` option of the browser context is false.

Usage

```
page.tap(selector)
page.tap(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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.

- `strict` bool *(optional)* Added in: v1.14

  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.

- `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

> **DISCOURAGED**
>
> Use locator-based locator.text_content() instead. Read more about locators.

Returns `element.textContent`.

Usage

```
page.text_content(selector)
page.text_content(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `strict` bool *(optional)* Added in: v1.14

  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|str

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().

Sends a `keydown`, `keypress`/`input`, and `keyup` event for each character in the text. `page.type` can be used to send fine-grained keyboard events. To fill values in form fields, use page.fill().

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

Usage

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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`.

- `strict` bool *(optional)* Added in: v1.14

  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.

uncheck

Added in: v1.8

> **DISCOURAGED**
>
> Use locator-based locator.uncheck() instead. Read more about locators.

This method unchecks an element matching `selector` by performing the following steps:

1. Find an element matching `selector`. If there is none, wait until a matching element is attached to the DOM.

2. Ensure that matched element is a checkbox or a radio input. If not, this method throws. If the element is already unchecked, 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 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

```
page.uncheck(selector)
page.uncheck(selector, **kwargs)
```

Arguments

- `selector` str

  A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

- `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.

- `strict` bool *(optional)* Added in: v1.14

  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.

- `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_selector

Added in: v1.8

> **DISCOURAGED**
>
> Use web assertions that assert visibility or a locator-based locator.wait_for() instead. Read more about locators.

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

> **NOTE**
>
> Playwright automatically waits for element to be ready before performing an action. Using Locator objects and web-first assertions makes the code wait-for-selector-free.

Wait for the `selector` 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

This method works across navigations:

Sync Async

```python
from playwright.sync_api import sync_playwright, Playwright

def run(playwright: Playwright):
    chromium = playwright.chromium
    browser = chromium.launch()
    page = browser.new_page()
    for current_url in ["https://google.com", "https://bbc.com"]:
        page.goto(current_url, wait_until="domcontentloaded")
        element = page.wait_for_selector("img")
        print("Loaded image: " + str(element.get_attribute("src")))
    browser.close()

with sync_playwright() as playwright:
    run(playwright)
```

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: v1.14

  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

wait_for_timeout

Added in: v1.8

> **DISCOURAGED**
>
> Never wait for timeout in production. Tests that wait for time are inherently flaky. Use Locator actions and web assertions that wait automatically.

Waits for the given `timeout` in milliseconds.

Note that `page.waitForTimeout()` should only be used for debugging. Tests using the timer in production are going to be flaky. Use signals such as network events, selectors becoming visible and others instead.

Usage

Sync | Async

```
# wait for 1 second
page.wait_for_timeout(1000)
```

Arguments

- `timeout` float

  A timeout to wait for