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:

```
const { webkit } = require('playwright');  // Or 'chromium' or 'firefox'.

(async () => {
  const browser = await webkit.launch();
  const context = await browser.newContext();
  const page = await context.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'screenshot.png'});
  await browser.close();
})();
```

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', () => console.log('Page loaded!'));
```

To unsubscribe from events use the `removeListener` method:

```
function logRequest(interceptedRequest) {
  console.log('A request was made:', interceptedRequest.url());
}
page.on('request', logRequest);
// Sometime later...
page.removeListener('request', logRequest);
```

- page.on('close')
- page.on('console')
- page.on('crash')

- page.on('dialog')
- page.on('domcontentloaded')
- page.on('download')
- page.on('filechooser')
- page.on('frameattached')
- page.on('framedetached')
- page.on('framenavigated')
- page.on('load')
- page.on('pageerror')
- page.on('popup')
- page.on('request')
- page.on('requestfailed')
- page.on('requestfinished')
- page.on('response')
- page.on('websocket')
- page.on('worker')
- page.$(selector[, options])
- page.$$(selector)
- page.$$eval(selector, pageFunction[, arg])
- page.$eval(selector, pageFunction[, arg, options])
- page.accessibility
- page.addInitScript(script[, arg])
- page.addScriptTag([options])
- page.addStyleTag([options])
- page.bringToFront()
- page.check(selector[, options])
- page.click(selector[, options])
- page.close([options])
- page.content()
- page.context()
- page.coverage
- page.dblclick(selector[, options])
- page.dispatchEvent(selector, type[, eventInit, options])
- page.dragAndDrop(source, target[, options])
- page.emulateMedia([options])
- page.evaluate(pageFunction[, arg])
- page.evaluateHandle(pageFunction[, arg])

- page.exposeBinding(name, callback[, options])
- page.exposeFunction(name, callback)
- page.fill(selector, value[, options])
- page.focus(selector[, options])
- page.frame(frameSelector)
- page.frameLocator(selector)
- page.frames()
- page.getAttribute(selector, name[, options])
- page.getByAltText(text[, options])
- page.getByLabel(text[, options])
- page.getByPlaceholder(text[, options])
- page.getByRole(role[, options])
- page.getByTestId(testId)
- page.getByText(text[, options])
- page.getByTitle(text[, options])
- page.goBack([options])
- page.goForward([options])
- page.goto(url[, options])
- page.hover(selector[, options])
- page.innerHTML(selector[, options])
- page.innerText(selector[, options])
- page.inputValue(selector[, options])
- page.isChecked(selector[, options])
- page.isClosed()
- page.isDisabled(selector[, options])
- page.isEditable(selector[, options])
- page.isEnabled(selector[, options])
- page.isHidden(selector[, options])
- page.isVisible(selector[, options])
- page.keyboard
- page.locator(selector[, options])
- page.mainFrame()
- page.mouse
- page.opener()
- page.pause()
- page.pdf([options])
- page.press(selector, key[, options])

- page.reload([options])
- page.request
- page.route(url, handler[, options])
- page.routeFromHAR(har[, options])
- page.screenshot([options])
- page.selectOption(selector, values[, options])
- page.setChecked(selector, checked[, options])
- page.setContent(html[, options])
- page.setDefaultNavigationTimeout(timeout)
- page.setDefaultTimeout(timeout)
- page.setExtraHTTPHeaders(headers)
- page.setInputFiles(selector, files[, options])
- page.setViewportSize(viewportSize)
- page.tap(selector[, options])
- page.textContent(selector[, options])
- page.title()
- page.touchscreen
- page.type(selector, text[, options])
- page.uncheck(selector[, options])
- page.unroute(url[, handler])
- page.url()
- page.video()
- page.viewportSize()
- page.waitForEvent(event[, optionsOrPredicate, options])
- page.waitForFunction(pageFunction[, arg, options])
- page.waitForLoadState([state, options])
- page.waitForNavigation([options])
- page.waitForRequest(urlOrPredicate[, options])
- page.waitForResponse(urlOrPredicate[, options])
- page.waitForSelector(selector[, options])
- page.waitForTimeout(timeout)
- page.waitForURL(url[, options])
- page.workers()

page.on('close')

Added in: v1.8

- type: <Page>

Emitted when the page closes.

page.on('console')

Added in: v1.8

- type: <ConsoleMessage>

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` appear as arguments on the event handler.

An example of handling `console` event:

```
page.on('console', async msg => {
  const values = [];
  for (const arg of msg.args())
    values.push(await arg.jsonValue());
  console.log(...values);
});
await page.evaluate(() => console.log('hello', 5, {foo: 'bar'}));
```

page.on('crash')

Added in: v1.8

- type: <Page>

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:

```
try {
  // Crash might happen during a click.
  await page.click('button');
  // Or while waiting for an event.
  await page.waitForEvent('popup');
} catch (e) {
  // When the page crashes, exception message contains 'crash'.
}
```

page.on('dialog')

Added in: v1.8

- type: <Dialog>

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

```
page.on('dialog', dialog => {
  dialog.accept();
});
```

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

page.on('domcontentloaded')

Added in: v1.9

- type: <Page>

Emitted when the JavaScript `DOMContentLoaded` event is dispatched.

page.on('download')

Added in: v1.8

- type: <Download>

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

page.on('filechooser')

Added in: v1.9

- type: <FileChooser>

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 fileChooser.setFiles(files[, options]) that can be uploaded after that.

```
page.on('filechooser', async (fileChooser) => {
  await fileChooser.setFiles('/tmp/myfile.pdf');
});
```

page.on('frameattached')

Added in: v1.9

- type: <Frame>

Emitted when a frame is attached.

page.on('framedetached')

Added in: v1.9

- type: <Frame>

Emitted when a frame is detached.

page.on('framenavigated')

Added in: v1.9

- type: <Frame>

Emitted when a frame is navigated to a new url.

page.on('load')

Added in: v1.8

- type: <Page>

Emitted when the JavaScript `load` event is dispatched.

page.on('pageerror')

Added in: v1.9

- type: <Error>

Emitted when an uncaught exception happens within the page.

```
// Log all uncaught errors to the terminal
page.on('pageerror', exception => {
  console.log(`Uncaught exception: "${exception}"`);
});

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

page.on('popup')

Added in: v1.8

- type: <Page>

Emitted when the page opens a new tab or window. This event is emitted in addition to the browserContext.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.

```
// Note that Promise.all prevents a race condition
// between evaluating and waiting for the popup.
const [popup] = await Promise.all([
  // It is important to call waitForEvent first.
  page.waitForEvent('popup'),
  // Opens the popup.
  page.evaluate(() => window.open('https://example.com')),
]);
console.log(await popup.evaluate('location.href'));
```

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

page.on('request')

Added in: v1.8

- type: <Request>

Emitted when a page issues a request. The request object is read-only. In order to intercept and mutate requests, see page.route(url, handler[, options]) or browserContext.route(url, handler[, options]).

page.on('requestfailed')

Added in: v1.9

- type: <Request>

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

```
page.on('requestfailed', request => {
  console.log(request.url() + ' ' + request.failure().errorText);
});
```

> **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.

page.on('requestfinished')

Added in: v1.9

- type: <Request>

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

page.on('response')

Added in: v1.8

- type: <Response>

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

page.on('websocket')

Added in: v1.9

- type: <WebSocket>

Emitted when WebSocket request is sent.

page.on('worker')

Added in: v1.8

- type: <Worker>

Emitted when a dedicated WebWorker is spawned by the page.

page.$(selector[, options])

Added in: v1.9

- `selector` <string> A selector to query for. See working with selectors for more details.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
- returns: <Promise<null|ElementHandle>>

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

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.waitFor([options]).

Shortcut for main frame's frame.$(selector[, options]).

page.$$(selector)

Added in: v1.9

- `selector` <string> A selector to query for. See working with selectors for more details.
- returns: <Promise<Array<ElementHandle>>>

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

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

Shortcut for main frame's frame.$$(selector).

page.$eval(selector, pageFunction[, arg, options])

Added in: v1.9

- `selector` <string> A selector to query for. See working with selectors for more details.
- `pageFunction` <function(Element)> Function to be evaluated in the page context.
- `arg?` <EvaluationArgument> Optional argument to pass to `pageFunction`.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
- returns: <Promise<Serializable>>

> **CAUTION**
>
> This method does not wait for the element to pass actionability checks and therefore can lead to the flaky tests. Use locator.evaluate(pageFunction[, arg, options]), 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 `pageFunction`. If no elements match the selector, the method throws an error. Returns the value of `pageFunction`.

If `pageFunction` returns a Promise, then page.$eval(selector, pageFunction[, arg, options]) would wait for the promise to resolve and return its value.

Examples:

```
const searchValue = await page.$eval('#search', el => el.value);
const preloadHref = await page.$eval('link[rel=preload]', el => el.href);
const html = await page.$eval('.main-container', (e, suffix) => e.outerHTML +
```

```
suffix, 'hello');
// In TypeScript, this example requires an explicit type annotation
(HTMLLinkElement) on el:
const preloadHrefTS = await page.$eval('link[rel=preload]', (el:
HTMLLinkElement) => el.href);
```

Shortcut for main frame's frame.$eval(selector, pageFunction[, arg, options]).

page.$$eval(selector, pageFunction[, arg])

Added in: v1.9

- `selector` <string> A selector to query for. See working with selectors for more details.
- `pageFunction` <function(Array<Element>)> Function to be evaluated in the page context.
- `arg?` <EvaluationArgument> Optional argument to pass to `pageFunction`.
- returns: <Promise<Serializable>>

> **NOTE**
>
> In most cases, locator.evaluateAll(pageFunction[, arg]), 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 `pageFunction`. Returns the result of `pageFunction` invocation.

If `pageFunction` returns a Promise, then page.$$eval(selector, pageFunction[, arg]) would wait for the promise to resolve and return its value.

Examples:

```
const divCounts = await page.$$eval('div', (divs, min) => divs.length >= min,
10);
```

page.addInitScript(script[, arg])

Added in: v1.8

- `script` <function|string|Object> Script to be evaluated in the page.

  - `path?` <string> Path to the JavaScript file. If `path` is a relative path, then it is resolved relative to the current working directory. Optional.
  - `content?` <string> Raw script content. Optional.
- `arg?` <Serializable> Optional argument to pass to `script` (only supported when passing a function).
- returns: <Promise<void>>

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`.

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

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

```
// In your playwright script, assuming the preload.js file is in same directory
await page.addInitScript({ path: './preload.js' });
```

> **NOTE**
>
> The order of evaluation of multiple scripts installed via browserContext.addInitScript(script[, arg]) and page.addInitScript(script[, arg]) is not defined.

page.addScriptTag([options])

Added in: v1.8

- `options?` <Object>
  - `content?` <string> Raw JavaScript content to be injected into frame.
  - `path?` <string> 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?` <string> Script type. Use 'module' in order to load a Javascript ES6 module. See script for more details.
  - `url?` <string> URL of a script to be added.

- returns: <Promise<ElementHandle>>

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.

Shortcut for main frame's frame.addScriptTag([options]).

page.addStyleTag([options])

Added in: v1.8

- `options?` <Object>
  - `content?` <string> Raw CSS content to be injected into frame.
  - `path?` <string> 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?` <string> URL of the `<link>` tag.
- returns: <Promise<ElementHandle>>

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.

Shortcut for main frame's frame.addStyleTag([options]).

page.bringToFront()

Added in: v1.8

- returns: <Promise<void>>

Brings page to front (activates tab).

page.check(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `force?` <boolean> Whether to bypass the actionability checks. Defaults to `false`.
  - `noWaitAfter?` <boolean> 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?` <Object> A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element. Added in: v1.11
    - `x` <number>
    - `y` <number>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `trial?` <boolean> 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. Added in: v1.11
- returns: <Promise<void>>

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 `noWaitAfter` 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.

Shortcut for main frame's frame.check(selector[, options]).

page.click(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more

details.

- `options?` <Object>
  - `button?` <"left"|"right"|"middle"> Defaults to `left`.
  - `clickCount?` <number> defaults to 1. See UIEvent.detail.
  - `delay?` <number> Time to wait between `mousedown` and `mouseup` in milliseconds. Defaults to 0.
  - `force?` <boolean> Whether to bypass the actionability checks. Defaults to `false`.
  - `modifiers?` <Array<"Alt"|"Control"|"Meta"|"Shift">> 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.
  - `noWaitAfter?` <boolean> 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?` <Object> A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.
    - `x` <number>
    - `y` <number>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `trial?` <boolean> 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. Added in: v1.11
- returns: <Promise<void>>

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 `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.

Shortcut for main frame's frame.click(selector[, options]).

page.close([options])

Added in: v1.8

- `options?` <Object>
  - `runBeforeUnload?` <boolean> Defaults to `false`. Whether to run the before unload page handlers.
- returns: <Promise<void>>

If `runBeforeUnload` is `false`, does not run any unload handlers and waits for the page to be closed. If `runBeforeUnload` 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 `runBeforeUnload` is passed as true, a `beforeunload` dialog might be summoned and should be handled manually via page.on('dialog') event.

page.content()

Added in: v1.8

- returns: <Promise<string>>

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

page.context()

Added in: v1.8

- returns: <BrowserContext>

Get the browser context that the page belongs to.

page.dblclick(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `button?` <"left"|"right"|"middle"> Defaults to `left`.
  - `delay?` <number> Time to wait between `mousedown` and `mouseup` in milliseconds. Defaults to 0.
  - `force?` <boolean> Whether to bypass the actionability checks. Defaults to `false`.
  - `modifiers?` <Array<"Alt"|"Control"|"Meta"|"Shift">> 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.
  - `noWaitAfter?` <boolean> 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?` <Object> A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.
    - `x` <number>
    - `y` <number>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `trial?` <boolean> 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. Added in: v1.11
- returns: <Promise<void>>

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 `noWaitAfter` 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.

Shortcut for main frame's frame.dblclick(selector[, options]).

page.dispatchEvent(selector, type[, eventInit, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `type` <string> DOM event type: `"click"`, `"dragstart"`, etc.
- `eventInit?` <EvaluationArgument> Optional event-specific initialization properties.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<void>>

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().

```
await page.dispatchEvent('button#submit', 'click');
```

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

Since `eventInit` 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:

```
// Note you can only create DataTransfer in Chromium and Firefox
const dataTransfer = await page.evaluateHandle(() => new DataTransfer());
await page.dispatchEvent('#source', 'dragstart', { dataTransfer });
```

page.dragAndDrop(source, target[, options])

Added in: v1.13

- `source` <string> A selector to search for an element to drag. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `target` <string> A selector to search for an element to drop onto. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `force?` <boolean> Whether to bypass the actionability checks. Defaults to `false`.
  - `noWaitAfter?` <boolean> 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`.
  - `sourcePosition?` <Object> 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. Added in: v1.14
    - `x` <number>
    - `y` <number>

- `strict?` <boolean> 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. Added in: v1.14
- `targetPosition?` <Object> 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. Added in: v1.14
  - `x` <number>
  - `y` <number>
- `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- `trial?` <boolean> 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.

- returns: <Promise<void>>

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`.

```
await page.dragAndDrop('#source', '#target');
// or specify exact positions relative to the top-left corners of the elements:
await page.dragAndDrop('#source', '#target', {
  sourcePosition: { x: 34, y: 7 },
  targetPosition: { x: 10, y: 20 },
});
```

page.emulateMedia([options])

Added in: v1.8

- `options?` <Object>
  - `colorScheme?` <null|"light"|"dark"|"no-preference"> Emulates `'prefers-colors-scheme'` media feature, supported values are `'light'`, `'dark'`, `'no-preference'`. Passing `null` disables color scheme emulation. Added in: v1.9
  - `forcedColors?` <null|"active"|"none"> Emulates `'forced-colors'` media feature, supported values are `'active'` and `'none'`. Passing `null` disables forced colors emulation. Added in: v1.15
  - `media?` <null|"screen"|"print"> Changes the CSS media type of the page. The only allowed values are `'screen'`, `'print'` and `null`. Passing `null` disables CSS media emulation. Added in: v1.9

  - `reducedMotion?` <null|"reduce"|"no-preference"> Emulates `'prefers-reduced-motion'` media feature, supported values are `'reduce'`, `'no-preference'`. Passing `null` disables reduced motion emulation. Added in: v1.12
- returns: <Promise<void>>

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

```
await page.evaluate(() => matchMedia('screen').matches);
// → true
await page.evaluate(() => matchMedia('print').matches);
// → false

await page.emulateMedia({ media: 'print' });
await page.evaluate(() => matchMedia('screen').matches);
// → false
await page.evaluate(() => matchMedia('print').matches);
// → true

await page.emulateMedia({});
await page.evaluate(() => matchMedia('screen').matches);
// → true
await page.evaluate(() => matchMedia('print').matches);
// → false
```

```
await page.emulateMedia({ colorScheme: 'dark' });
await page.evaluate(() => matchMedia('(prefers-color-scheme: dark)').matches);
// → true
await page.evaluate(() => matchMedia('(prefers-color-scheme: light)').matches);
// → false
await page.evaluate(() => matchMedia('(prefers-color-scheme: no-preference)').matches);
// → false
```

page.evaluate(pageFunction[, arg])

Added in: v1.8

- `pageFunction` <function|string> Function to be evaluated in the page context.
- `arg?` <EvaluationArgument> Optional argument to pass to `pageFunction`.
- returns: <Promise<Serializable>>

Returns the value of the `pageFunction` invocation.

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

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

Passing argument to `pageFunction`:

```
const result = await page.evaluate(([x, y]) => {
  return Promise.resolve(x * y);
}, [7, 8]);
console.log(result); // prints "56"
```

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

```
console.log(await page.evaluate('1 + 2')); // prints "3"
const x = 10;
console.log(await page.evaluate(`1 + ${x}`)); // prints "11"
```

ElementHandle instances can be passed as an argument to the page.evaluate(pageFunction[, arg]):

```
const bodyHandle = await page.evaluate('document.body');
const html = await page.evaluate(([body, suffix]) => body.innerHTML + suffix,
[bodyHandle, 'hello']);
await bodyHandle.dispose();
```

Shortcut for main frame's frame.evaluate(pageFunction[, arg]).

page.evaluateHandle(pageFunction[, arg])

Added in: v1.8

- `pageFunction` <function|string> Function to be evaluated in the page context.
- `arg?` <EvaluationArgument> Optional argument to pass to `pageFunction`.
- returns: <Promise<JSHandle>>

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

The only difference between page.evaluate(pageFunction[, arg]) and page.evaluateHandle(pageFunction[, arg]) is that page.evaluateHandle(pageFunction[, arg]) returns JSHandle.

If the function passed to the page.evaluateHandle(pageFunction[, arg]) returns a Promise, then page.evaluateHandle(pageFunction[, arg]) would wait for the promise to resolve and return its value.

```
const aWindowHandle = await page.evaluateHandle(() => Promise.resolve(window));
aWindowHandle; // Handle for the window object.
```

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

```
const aHandle = await page.evaluateHandle('document'); // Handle for the
'document'
```

JSHandle instances can be passed as an argument to the page.evaluateHandle(pageFunction[, arg]):

```
const aHandle = await page.evaluateHandle(() => document.body);
const resultHandle = await page.evaluateHandle(body => body.innerHTML,
aHandle);
console.log(await resultHandle.jsonValue());
await resultHandle.dispose();
```

page.exposeBinding(name, callback[, options])

Added in: v1.8

- `name` <string> Name of the function on the window object.
- `callback` <function> Callback function that will be called in the Playwright's context.
- `options?` <Object>
  - `handle?` <boolean> 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.
- returns: <Promise<void>>

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 browserContext.exposeBinding(name, callback[, options]) for the context-wide version.

> **NOTE**
>
> Functions installed via page.exposeBinding(name, callback[, options]) survive navigations.

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

```
const { webkit } = require('playwright');  // Or 'chromium' or 'firefox'.

(async () => {
  const browser = await webkit.launch({ headless: false });
  const context = await browser.newContext();
  const page = await context.newPage();
  await page.exposeBinding('pageURL', ({ page }) => page.url());
  await page.setContent(`
    <script>
      async function onClick() {
        document.querySelector('div').textContent = await window.pageURL();
      }
    </script>
    <button onclick="onClick()">Click me</button>
    <div></div>
  `);
  await page.click('button');
})();
```

An example of passing an element handle:

```
await page.exposeBinding('clicked', async (source, element) => {
  console.log(await element.textContent());
}, { handle: true });
await page.setContent(`
  <script>
    document.addEventListener('click', event => window.clicked(event.target));
  </script>
  <div>Click me</div>
  <div>Or click me</div>
`);
```

page.exposeFunction(name, callback)

Added in: v1.8

- `name` <string> Name of the function on the window object
- `callback` <function> Callback function which will be called in Playwright's context.
- returns: <Promise<void>>

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 browserContext.exposeFunction(name, callback) for context-wide exposed function.

> **NOTE**
>
> Functions installed via page.exposeFunction(name, callback) survive navigations.

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

```
const { webkit } = require('playwright');  // Or 'chromium' or 'firefox'.
const crypto = require('crypto');

(async () => {
  const browser = await webkit.launch({ headless: false });
  const page = await browser.newPage();
  await page.exposeFunction('sha256', text =>
crypto.createHash('sha256').update(text).digest('hex'));
  await page.setContent(`
    <script>
      async function onClick() {
        document.querySelector('div').textContent = await
window.sha256('PLAYWRIGHT');
      }
    </script>
    <button onclick="onClick()">Click me</button>
    <div></div>
  `);
  await page.click('button');
})();
```

page.fill(selector, value[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `value` <string> Value to fill for the `<input>`, `<textarea>` or `[contenteditable]` element.
- `options?` <Object>
  - `force?` <boolean> Whether to bypass the actionability checks. Defaults to `false`. Added in: v1.13
  - `noWaitAfter?` <boolean> 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?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<void>>

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 page.type(selector, text[, options]).

Shortcut for main frame's frame.fill(selector, value[, options]).

page.focus(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>

  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<void>>

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.

Shortcut for main frame's frame.focus(selector[, options]).

page.frame(frameSelector)

Added in: v1.8

- `frameSelector` <string|Object> Frame name or other frame lookup options.
  - `name?` <string> Frame name specified in the `iframe`'s `name` attribute. Optional.
  - `url?` <string|RegExp|function(URL):boolean> A glob pattern, regex pattern or predicate receiving frame's `url` as a URL object. Optional.
- returns: <null|Frame>

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

```
const frame = page.frame('frame-name');
```

```
const frame = page.frame({ url: /.*domain.*/ });
```

page.frameLocator(selector)

Added in: v1.17

- `selector` <string> A selector to use when resolving DOM element. See working with selectors for more details.
- returns: <FrameLocator>

When working with iframes, you can create a frame locator that will enter the iframe and allow selecting elements in that iframe. Following snippet locates element with text "Submit" in the iframe with id `my-frame`, like `<iframe id="my-frame">`:

```
const locator = page.frameLocator('#my-iframe').getByText('Submit');
await locator.click();
```

page.frames()

Added in: v1.8

- returns: <Array<Frame>>

An array of all frames attached to the page.

page.getAttribute(selector, name[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `name` <string> Attribute name to get the value for.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<null|string>>

Returns element attribute value.

page.getByAltText(text[, options])

Added in: v1.27

- `text` <string|RegExp> Text to locate the element for.
- `options?` <Object>
  - `exact?` <boolean> Whether to find an exact match: case-sensitive and whole-string. Default to false. Ignored when locating by a regular expression.
- returns: <Locator>

Allows locating elements by their alt text. For example, this method will find the image by alt text "Castle":

```
<img alt='Castle'>
```

page.getByLabel(text[, options])

Added in: v1.27

- `text` <string|RegExp> Text to locate the element for.
- `options?` <Object>
  - `exact?` <boolean> Whether to find an exact match: case-sensitive and whole-string. Default to false. Ignored when locating by a regular expression.
- returns: <Locator>

Allows locating input elements by the text of the associated label. For example, this method will find the input by label text Password in the following DOM:

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

page.getByPlaceholder(text[, options])

Added in: v1.27

- `text` <string|RegExp> Text to locate the element for.
- `options?` <Object>
  - `exact?` <boolean> Whether to find an exact match: case-sensitive and whole-string. Default to false. Ignored when locating by a regular expression.
- returns: <Locator>

Allows locating input elements by the placeholder text. For example, this method will find the input by placeholder "Country":

```
<input placeholder="Country">
```

page.getByRole(role[, options])

Added in: v1.27

- `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.

- `options?` <Object>
  - `checked?` <boolean> An attribute that is usually set by `aria-checked` or native `<input type=checkbox>` controls. Available values for checked are `true`, `false` and `"mixed"`.

    Learn more about `aria-checked`.

  - `disabled?` <boolean> A boolean 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`.

  - `expanded?` <boolean> A boolean attribute that is usually set by `aria-expanded`.

    Learn more about `aria-expanded`.

  - `includeHidden?` <boolean> A boolean attribute 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?` <number> 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?` <string|RegExp> A string attribute that matches accessible name.

Learn more about accessible name.

- `pressed?` <boolean> An attribute that is usually set by `aria-pressed`. Available values for pressed are `true`, `false` and `"mixed"`.

  Learn more about `aria-pressed`.

- `selected?` <boolean> A boolean attribute that is usually set by `aria-selected`.

  Learn more about `aria-selected`.

- returns: <Locator>

Allows locating elements by their ARIA role, ARIA attributes and accessible name. Note that role selector **does not replace** accessibility audits and conformance tests, but rather gives early feedback about the ARIA guidelines.

Note that 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.

page.getByTestId(testId)

Added in: v1.27

- `testId` <string> Id to locate the element by.
- returns: <Locator>

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

page.getByText(text[, options])

Added in: v1.27

- `text` <string|RegExp> Text to locate the element for.
- `options?` <Object>
  - `exact?` <boolean> Whether to find an exact match: case-sensitive and whole-string. Default to false. Ignored when locating by a regular expression.
- returns: <Locator>

Allows locating elements that contain given text.

page.getByTitle(text[, options])

Added in: v1.27

- `text` <string|RegExp> Text to locate the element for.
- `options?` <Object>
  - `exact?` <boolean> Whether to find an exact match: case-sensitive and whole-string. Default to false. Ignored when locating by a regular expression.
- returns: <Locator>

Allows locating elements by their title. For example, this method will find the button by its title "Submit":

```
<button title='Place the order'>Order Now</button>
```

page.goBack([options])

Added in: v1.8

- `options?` <Object>
  - `timeout?` <number> Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultNavigationTimeout(timeout), browserContext.setDefaultTimeout(timeout), page.setDefaultNavigationTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `waitUntil?` <"load"|"domcontentloaded"|"networkidle"|"commit"> 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'` - consider operation to be finished when there are no network connections for at least `500` ms.
    - `'commit'` - consider operation to be finished when network response is received and the document started loading.
- returns: <Promise<null|Response>>

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.

page.goForward([options])

Added in: v1.8

- `options?` <Object>
  - `timeout?` <number> Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultNavigationTimeout(timeout), browserContext.setDefaultTimeout(timeout), page.setDefaultNavigationTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `waitUntil?` <"load"|"domcontentloaded"|"networkidle"|"commit"> 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'` - consider operation to be finished when there are no network connections for at least `500` ms.
    - `'commit'` - consider operation to be finished when network response is received and the document started loading.
- returns: <Promise<null|Response>>

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.

page.goto(url[, options])

Added in: v1.8

- `url` <string> URL to navigate page to. The url should include scheme, e.g. `https://`. When a `baseURL` via the context options was provided and the passed URL is a path, it gets merged via the `new URL()` constructor.
- `options?` <Object>
  - `referer?` <string> Referer header value. If provided it will take preference over the referer header value set by page.setExtraHTTPHeaders(headers).
  - `timeout?` <number> Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultNavigationTimeout(timeout), browserContext.setDefaultTimeout(timeout),

page.setDefaultNavigationTimeout(timeout) or page.setDefaultTimeout(timeout) methods.

- `waitUntil?` <"load"|"domcontentloaded"|"networkidle"|"commit"> 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'` - consider operation to be finished when there are no network connections for at least `500` ms.
  - `'commit'` - consider operation to be finished when network response is received and the document started loading.

- returns: <Promise<null|Response>>

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.

Shortcut for main frame's frame.goto(url[, options])

page.hover(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `force?` <boolean> Whether to bypass the actionability checks. Defaults to `false`.
  - `modifiers?` <Array<"Alt"|"Control"|"Meta"|"Shift">> 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.
  - `position?` <Object> A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.
    - `x` <number>
    - `y` <number>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `trial?` <boolean> 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. Added in: v1.11
- returns: <Promise<void>>

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.

Shortcut for main frame's frame.hover(selector[, options]).

page.innerHTML(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<string>>

Returns `element.innerHTML`.

page.innerText(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<string>>

Returns `element.innerText`.

page.inputValue(selector[, options])

Added in: v1.13

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<string>>

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.

page.isChecked(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<boolean>>

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

page.isClosed()

Added in: v1.8

- returns: <boolean>

Indicates that the page has been closed.

page.isDisabled(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<boolean>>

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

page.isEditable(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<boolean>>

Returns whether the element is editable.

page.isEnabled(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<boolean>>

Returns whether the element is enabled.

page.isHidden(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> **DEPRECATED** This option is ignored. page.isHidden(selector[, options]) does not wait for the element to become hidden and returns immediately.
- returns: <Promise<boolean>>

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

page.isVisible(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more

details.

- `options?` <Object>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> **DEPRECATED** This option is ignored. page.isVisible(selector[, options]) does not wait for the element to become visible and returns immediately.
- returns: <Promise<boolean>>

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

page.locator(selector[, options])

Added in: v1.14

- `selector` <string> A selector to use when resolving DOM element. See working with selectors for more details.
- `options?` <Object>
  - `has?` <Locator> 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.
  - `hasText?` <string|RegExp> 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>

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.

page.mainFrame()

Added in: v1.8

- returns: <Frame>

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

page.opener()

Added in: v1.8

- returns: <Promise<null|Page>>

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

page.pause()

Added in: v1.9

- returns: <Promise<void>>

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 browserType.launch([options]).

page.pdf([options])

Added in: v1.8

- `options?` <Object>
  - `displayHeaderFooter?` <boolean> Display header and footer. Defaults to `false`.
  - `footerTemplate?` <string> HTML template for the print footer. Should use the same format as the `headerTemplate`.

- `format?` <string> Paper format. If set, takes priority over `width` or `height` options. Defaults to 'Letter'.
- `headerTemplate?` <string> 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?` <string|number> Paper height, accepts values labeled with units.
- `landscape?` <boolean> Paper orientation. Defaults to `false`.
- `margin?` <Object> Paper margins, defaults to none.
  - `top?` <string|number> Top margin, accepts values labeled with units. Defaults to `0`.
  - `right?` <string|number> Right margin, accepts values labeled with units. Defaults to `0`.
  - `bottom?` <string|number> Bottom margin, accepts values labeled with units. Defaults to `0`.
  - `left?` <string|number> Left margin, accepts values labeled with units. Defaults to `0`.
- `pageRanges?` <string> Paper ranges to print, e.g., '1-5, 8, 11-13'. Defaults to the empty string, which means print all pages.
- `path?` <string> 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.
- `preferCSSPageSize?` <boolean> 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.
- `printBackground?` <boolean> Print background graphics. Defaults to `false`.
- `scale?` <number> Scale of the webpage rendering. Defaults to `1`. Scale amount must be between 0.1 and 2.
- `width?` <string|number> Paper width, accepts values labeled with units.

- returns: <Promise<Buffer>>

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.emulateMedia([options]) 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.

```
// Generates a PDF with 'screen' media type.
await page.emulateMedia({media: 'screen'});
await 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**
>
> `headerTemplate` and `footerTemplate` markup have the following limitations: > 1. Script tags inside templates are not evaluated. > 2. Page styles are not visible inside templates.

page.press(selector, key[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `key` <string> Name of the key to press or a character to generate, such as `ArrowLeft` or `a`.
- `options?` <Object>
  - `delay?` <number> Time to wait between `keydown` and `keyup` in milliseconds. Defaults to 0.
  - `noWaitAfter?` <boolean> 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?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<void>>

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

`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.

```
const page = await browser.newPage();
await page.goto('https://keycode.info');
await page.press('body', 'A');
await page.screenshot({ path: 'A.png' });
await page.press('body', 'ArrowLeft');
await page.screenshot({ path: 'ArrowLeft.png' });
await page.press('body', 'Shift+O');
await page.screenshot({ path: 'O.png' });
await browser.close();
```

page.reload([options])

Added in: v1.8

- `options?` <Object>
  - `timeout?` <number> Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultNavigationTimeout(timeout), browserContext.setDefaultTimeout(timeout), page.setDefaultNavigationTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `waitUntil?` <"load"|"domcontentloaded"|"networkidle"|"commit"> 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'` - consider operation to be finished when there are no network connections for at least `500` ms.
    - `'commit'` - consider operation to be finished when network response is received and the document started loading.
- returns: <Promise<null|Response>>

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.

page.route(url, handler[, options])

Added in: v1.8

- `url` <string|RegExp|function(URL):boolean> A glob pattern, regex pattern or predicate receiving URL to match while routing. When a `baseURL` via the context options was provided and the passed URL is a path, it gets merged via the `new URL()` constructor.
- `handler` <function(Route, Request)> handler function to route the request.
- `options?` <Object>
  - `times?` <number> How often a route should be used. By default it will be used every time. Added in: v1.15
- returns: <Promise<void>>

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(url, handler[, options]) will not intercept requests intercepted by Service Worker. See this issue. We recommend disabling Service Workers when using request interception by setting `Browser.newContext.serviceWorkers` to `'block'`.

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

```
const page = await browser.newPage();
await page.route('**/*.{png,jpg,jpeg}', route => route.abort());
await page.goto('https://example.com');
await browser.close();
```

or the same snippet using a regex pattern instead:

```
const page = await browser.newPage();
await page.route(/(\.png$)|(\.jpg$)/, route => route.abort());
await page.goto('https://example.com');
await 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:

```
await page.route('/api/**', route => {
  if (route.request().postData().includes('my-string'))
    route.fulfill({ body: 'mocked-data' });
  else
    route.continue();
});
```

Page routes take precedence over browser context routes (set up with browserContext.route(url, handler[, options])) when request matches both handlers.

To remove a route with its handler you can use page.unroute(url[, handler]).

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

page.routeFromHAR(har[, options])

Added in: v1.23

- `har` <string> 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.
- `options?` <Object>
  - `notFound?` <"abort"|"fallback"> 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?` <boolean> If specified, updates the given HAR with the actual network information instead of serving from file.
  - `url?` <string|RegExp> 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.
- returns: <Promise<void>>

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.newContext.serviceWorkers` to `'block'`.

page.screenshot([options])

Added in: v1.8

- `options?` <Object>
  - `animations?` <"disabled"|"allow"> 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"> When set to `"hide"`, screenshot will hide text caret. When set to `"initial"`, text caret behavior will not be changed. Defaults to `"hide"`.
  - `clip?` <Object> An object which specifies clipping of the resulting image. Should have the following fields:
    - `x` <number> x-coordinate of top-left corner of clip area
    - `y` <number> y-coordinate of top-left corner of clip area
    - `width` <number> width of clipping area
    - `height` <number> height of clipping area
  - `fullPage?` <boolean> When true, takes a screenshot of the full scrollable page, instead of the currently visible viewport. Defaults to `false`.
  - `mask?` <Array<Locator>> Specify locators that should be masked when the screenshot is taken. Masked elements will be overlaid with a pink box `#FF00FF` that completely covers its bounding box.
  - `omitBackground?` <boolean> Hides default white background and allows capturing screenshots with transparency. Not applicable to `jpeg` images. Defaults to `false`.

- `path?` <string> 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?` <number> The quality of the image, between 0-100. Not applicable to `png` images.

- `scale?` <"css"|"device"> 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 screenhots of high-dpi devices will be twice as large or even larger.

  Defaults to `"device"`.

- `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.

- `type?` <"png"|"jpeg"> Specify screenshot type, defaults to `png`.

- returns: <Promise<Buffer>>

Returns the buffer with the captured screenshot.

page.selectOption(selector, values[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `values` <null|string|ElementHandle|Array<string>|Object|Array<ElementHandle>|Array<Object>> Options to select. If the `<select>` has the `multiple` attribute, all matching options are selected, otherwise only the first option matching one of the passed options is selected. String values are equivalent to `{value:'string'}`. Option is considered matching if all specified properties match.
  - `value?` <string> Matches by `option.value`. Optional.
  - `label?` <string> Matches by `option.label`. Optional.
  - `index?` <number> Matches by the index. Optional.

- `options?` <Object>
  - `force?` <boolean> Whether to bypass the actionability checks. Defaults to `false`.
    Added in: v1.13
  - `noWaitAfter?` <boolean> 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?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<Array<string>>>

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.

```
// single selection matching the value
page.selectOption('select#colors', 'blue');

// single selection matching the label
page.selectOption('select#colors', { label: 'Blue' });

// multiple selection
page.selectOption('select#colors', ['red', 'green', 'blue']);
```

Shortcut for main frame's frame.selectOption(selector, values[, options]).

page.setChecked(selector, checked[, options])

Added in: v1.15

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `checked` <boolean> Whether to check or uncheck the checkbox.
- `options?` <Object>
  - `force?` <boolean> Whether to bypass the actionability checks. Defaults to `false`.
  - `noWaitAfter?` <boolean> 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?` <Object> A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.
    - `x` <number>
    - `y` <number>
  - `strict?` <boolean> 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?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `trial?` <boolean> 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.
- returns: <Promise<void>>

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 `noWaitAfter` 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.

Shortcut for main frame's frame.setChecked(selector, checked[, options]).

page.setContent(html[, options])

Added in: v1.8

- `html` <string> HTML markup to assign to the page.
- `options?` <Object>
  - `timeout?` <number> Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultNavigationTimeout(timeout), browserContext.setDefaultTimeout(timeout), page.setDefaultNavigationTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `waitUntil?` <"load"|"domcontentloaded"|"networkidle"|"commit"> 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'` - consider operation to be finished when there are no network connections for at least `500` ms.
    - `'commit'` - consider operation to be finished when network response is received and the document started loading.
- returns: <Promise<void>>

page.setDefaultNavigationTimeout(timeout)

Added in: v1.8

- `timeout` <number> Maximum navigation time in milliseconds
- returns: <void>

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

- page.goBack([options])
- page.goForward([options])

- page.goto(url[, options])
- page.reload([options])
- page.setContent(html[, options])
- page.waitForNavigation([options])
- page.waitForURL(url[, options])

> **NOTE**
>
> page.setDefaultNavigationTimeout(timeout) takes priority over page.setDefaultTimeout(timeout), browserContext.setDefaultTimeout(timeout) and browserContext.setDefaultNavigationTimeout(timeout).

page.setDefaultTimeout(timeout)

Added in: v1.8

- `timeout` <number> Maximum time in milliseconds
- returns: <void>

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

> **NOTE**
>
> page.setDefaultNavigationTimeout(timeout) takes priority over page.setDefaultTimeout(timeout).

page.setExtraHTTPHeaders(headers)

Added in: v1.8

- `headers` <Object<string, string>> An object containing additional HTTP headers to be sent with every request. All header values must be strings.
- returns: <Promise<void>>

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

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

page.setInputFiles(selector, files[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `files` <string|Array<string>|Object|Array<Object>>
  - `name` <string> File name
  - `mimeType` <string> File type
  - `buffer` <Buffer> File content
- `options?` <Object>
  - `noWaitAfter?` <boolean> 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?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<void>>

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.

page.setViewportSize(viewportSize)

Added in: v1.8

- `viewportSize` <Object>
  - `width` <number> page width in pixels.
  - `height` <number> page height in pixels.
- returns: <Promise<void>>

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

page.setViewportSize(viewportSize) 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.setViewportSize(viewportSize) will also reset `screen` size, use browser.newContext([options]) with `screen` and `viewport` parameters if you need better control of these properties.

```
const page = await browser.newPage();
await page.setViewportSize({
  width: 640,
  height: 480,
});
await page.goto('https://example.com');
```

page.tap(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `force?` <boolean> Whether to bypass the actionability checks. Defaults to `false`.
  - `modifiers?` <Array<"Alt"|"Control"|"Meta"|"Shift">> 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.
  - `noWaitAfter?` <boolean> 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?` <Object> A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element.
    - `x` <number>
    - `y` <number>
  - `strict?` <boolean> 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. Added in: v1.14

    - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
    - `trial?` <boolean> 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. Added in: v1.11
- returns: <Promise<void>>

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 `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.

> **NOTE**
>
> page.tap(selector[, options]) requires that the `hasTouch` option of the browser context be set to true.

Shortcut for main frame's frame.tap(selector[, options]).

page.textContent(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
    - `strict?` <boolean> 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. Added in: v1.14
    - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the

browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.

- returns: <Promise<null|string>>

Returns `element.textContent`.

page.title()

Added in: v1.8

- returns: <Promise<string>>

Returns the page's title. Shortcut for main frame's frame.title().

page.type(selector, text[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `text` <string> A text to type into a focused element.
- `options?` <Object>
  - `delay?` <number> Time to wait between key presses in milliseconds. Defaults to 0.
  - `noWaitAfter?` <boolean> 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?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<void>>

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(selector, value[, options]).

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

```
await page.type('#mytextarea', 'Hello'); // Types instantly
await page.type('#mytextarea', 'World', {delay: 100}); // Types slower, like a user
```

Shortcut for main frame's frame.type(selector, text[, options]).

page.uncheck(selector[, options])

Added in: v1.8

- `selector` <string> A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used. See working with selectors for more details.
- `options?` <Object>
  - `force?` <boolean> Whether to bypass the actionability checks. Defaults to `false`.
  - `noWaitAfter?` <boolean> 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?` <Object> A point to use relative to the top-left corner of element padding box. If not specified, uses some visible point of the element. Added in: v1.11
    - `x` <number>
    - `y` <number>
  - `strict?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `trial?` <boolean> 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. Added in: v1.11
- returns: <Promise<void>>

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 `noWaitAfter` 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.

Shortcut for main frame's frame.uncheck(selector[, options]).

page.unroute(url[, handler])

Added in: v1.8

- `url` <string|RegExp|function(URL):boolean> A glob pattern, regex pattern or predicate receiving URL to match while routing.
- `handler?` <function(Route, Request)> Optional handler function to route the request.
- returns: <Promise<void>>

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

page.url()

Added in: v1.8

- returns: <string>

Shortcut for main frame's frame.url().

page.video()

Added in: v1.8

- returns: <null|Video>

Video object associated with this page.

page.viewportSize()

Added in: v1.8

- returns: <null|Object>
  - `width` <number> page width in pixels.
  - `height` <number> page height in pixels.

page.waitForEvent(event[, optionsOrPredicate, options])

Added in: v1.8

- `event` <string> Event name, same one typically passed into `*.on(event)`.
- `optionsOrPredicate?` <function|Object> Either a predicate that receives an event or an options object. Optional.
  - `predicate` <function> receives the event data and resolves to truthy value when the waiting should resolve.
  - `timeout?` <number> 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 browserContext.setDefaultTimeout(timeout).
- `options?` <Object>
  - `predicate?` <function> Receives the event data and resolves to truthy value when the waiting should resolve.
- returns: <Promise<Object>>

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.

```
// Note that Promise.all prevents a race condition
// between clicking and waiting for the event.
const [frame, _] = await Promise.all([
  // It is important to call waitForEvent before click to set up waiting.
  page.waitForEvent('framenavigated'),
  // Triggers the navigation.
  page.getByRole('button').click(),
]);
```

page.waitForFunction(pageFunction[, arg, options])

Added in: v1.8

- `pageFunction` <function|string> Function to be evaluated in the page context.
- `arg?` <EvaluationArgument> Optional argument to pass to `pageFunction`.
- `options?` <Object>
  - `polling?` <number|"raf"> If `polling` is `'raf'`, then `pageFunction` 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?` <number> 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 browserContext.setDefaultTimeout(timeout).
- returns: <Promise<JSHandle>>

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

The page.waitForFunction(pageFunction[, arg, options]) can be used to observe viewport size change:

```
const { webkit } = require('playwright');  // Or 'chromium' or 'firefox'.

(async () => {
  const browser = await webkit.launch();
  const page = await browser.newPage();
  const watchDog = page.waitForFunction(() => window.innerWidth < 100);
  await page.setViewportSize({width: 50, height: 50});
  await watchDog;
  await browser.close();
})();
```

To pass an argument to the predicate of page.waitForFunction(pageFunction[, arg, options]) function:

```
const selector = '.foo';
await page.waitForFunction(selector => !!document.querySelector(selector),
selector);
```

Shortcut for main frame's frame.waitForFunction(pageFunction[, arg, options]).

page.waitForLoadState([state, options])

Added in: v1.8

- `state?` <"load"|"domcontentloaded"|"networkidle"> 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'` - wait until there are no network connections for at least `500` ms.
- `options?` <Object>
  - `timeout?` <number> Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultNavigationTimeout(timeout), browserContext.setDefaultTimeout(timeout), page.setDefaultNavigationTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<void>>

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.

```
await page.getByRole('button').click(); // Click triggers navigation.
await page.waitForLoadState(); // The promise resolves after 'load' event.
```

```
const [popup] = await Promise.all([
  // It is important to call waitForEvent before click to set up waiting.
  page.waitForEvent('popup'),
  // Click triggers a popup.
  page.getByRole('button').click(),
])
await popup.waitForLoadState('domcontentloaded'); // The promise resolves after
'domcontentloaded' event.
console.log(await popup.title()); // Popup is ready to use.
```

Shortcut for main frame's frame.waitForLoadState([state, options]).

page.waitForNavigation([options])

Added in: v1.8

- `options?` <Object>

- `timeout?` <number> Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultNavigationTimeout(timeout), browserContext.setDefaultTimeout(timeout), page.setDefaultNavigationTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- `url?` <string|RegExp|function(URL):boolean> 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.
- `waitUntil?` <"load"|"domcontentloaded"|"networkidle"|"commit"> 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'` - consider operation to be finished when there are no network connections for at least `500` ms.
  - `'commit'` - consider operation to be finished when network response is received and the document started loading.

- returns: <Promise<null|Response>>

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`.

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:

```
// Note that Promise.all prevents a race condition
// between clicking and waiting for the navigation.
const [response] = await Promise.all([
  // It is important to call waitForNavigation before click to set up waiting.
  page.waitForNavigation(),
  // Clicking the link will indirectly cause a navigation.
  page.locator('a.delayed-navigation').click(),
]);
```

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

Shortcut for main frame's frame.waitForNavigation([options]).

page.waitForRequest(urlOrPredicate[, options])

Added in: v1.8

- `urlOrPredicate` <string|RegExp|function(Request):boolean|Promise<boolean>> Request URL string, regex or predicate receiving Request object.
- `options?` <Object>
  - `timeout?` <number> 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.setDefaultTimeout(timeout) method.
- returns: <Promise<Request>>

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

```
// Note that Promise.all prevents a race condition
// between clicking and waiting for the request.
const [request] = await Promise.all([
  // Waits for the next request with the specified url
  page.waitForRequest('https://example.com/resource'),
  // Triggers the request
  page.click('button.triggers-request'),
]);

// Alternative way with a predicate.
const [request] = await Promise.all([
  // Waits for the next request matching some conditions
  page.waitForRequest(request => request.url() === 'https://example.com' &&
request.method() === 'GET'),
  // Triggers the request
  page.click('button.triggers-request'),
]);
```

page.waitForResponse(urlOrPredicate[, options])

Added in: v1.8

- `urlOrPredicate` <string|RegExp|function(Response):boolean|Promise<boolean>> Request URL string, regex or predicate receiving Response object. When a `baseURL` via

the context options was provided and the passed URL is a path, it gets merged via the `new URL()` constructor.

- `options?` <Object>
  - `timeout?` <number> Maximum wait time in milliseconds, defaults to 30 seconds, pass `0` to disable the timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<Response>>

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

```
// Note that Promise.all prevents a race condition
// between clicking and waiting for the response.
const [response] = await Promise.all([
  // Waits for the next response with the specified url
  page.waitForResponse('https://example.com/resource'),
  // Triggers the response
  page.click('button.triggers-response'),
]);

// Alternative way with a predicate.
const [response] = await Promise.all([
  // Waits for the next response matching some conditions
  page.waitForResponse(response => response.url() === 'https://example.com' &&
response.status() === 200),
  // Triggers the response
  page.click('button.triggers-response'),
]);
```

page.waitForSelector(selector[, options])

Added in: v1.8

- `selector` <string> A selector to query for. See working with selectors for more details.
- `options?` <Object>
  - `state?` <"attached"|"detached"|"visible"|"hidden"> 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?` <boolean> 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. Added in: v1.14
  - `timeout?` <number> Maximum time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
- returns: <Promise<null|ElementHandle>>

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.

This method works across navigations:

```
const { chromium } = require('playwright');  // Or 'firefox' or 'webkit'.

(async () => {
  const browser = await chromium.launch();
  const page = await browser.newPage();
  for (let currentURL of ['https://google.com', 'https://bbc.com']) {
    await page.goto(currentURL);
    const element = await page.waitForSelector('img');
    console.log('Loaded image: ' + await element.getAttribute('src'));
  }
  await browser.close();
})();
```

page.waitForTimeout(timeout)

Added in: v1.8

- `timeout` <number> A timeout to wait for
- returns: <Promise<void>>

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.

```
// wait for 1 second
await page.waitForTimeout(1000);
```

Shortcut for main frame's frame.waitForTimeout(timeout).

page.waitForURL(url[, options])

Added in: v1.11

- `url` <string|RegExp|function(URL):boolean> 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.
- `options?` <Object>
  - `timeout?` <number> Maximum operation time in milliseconds, defaults to 30 seconds, pass `0` to disable timeout. The default value can be changed by using the browserContext.setDefaultNavigationTimeout(timeout), browserContext.setDefaultTimeout(timeout), page.setDefaultNavigationTimeout(timeout) or page.setDefaultTimeout(timeout) methods.
  - `waitUntil?` <"load"|"domcontentloaded"|"networkidle"|"commit"> 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'` - consider operation to be finished when there are no network connections for at least `500` ms.
    - `'commit'` - consider operation to be finished when network response is received and the document started loading.
- returns: <Promise<void>>

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

```
await page.click('a.delayed-navigation'); // Clicking the link will indirectly cause a navigation
await page.waitForURL('**/target.html');
```

Shortcut for main frame's frame.waitForURL(url[, options]).

page.workers()

Added in: v1.8

- returns: <Array<Worker>>

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

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

page.accessibility

Added in: v1.8

- type: <Accessibility>

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

page.coverage

Added in: v1.8

- type: <Coverage>

> **NOTE**
>
> Only available for Chromium atm.

Browser-specific Coverage implementation. See Coverage for more details.

page.keyboard

Added in: v1.8

- type: <Keyboard>

page.mouse

Added in: v1.8

- type: <Mouse>

page.request

Added in: v1.16

- type: <APIRequestContext>

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

page.touchscreen

Added in: v1.8

- type: <Touchscreen>