

[Home](#) > [puppeteer](#) > [Page](#)

Page class

Page provides methods to interact with a single tab or [extension background page](#) in Chromium.

Signature:

```
export declare class Page extends EventEmitter
```

Extends: [EventEmitter](#)

Remarks

One Browser instance might have multiple Page instances.

The constructor for this class is marked as internal. Third-party code should not call the constructor directly or create subclasses that extend the `Page` class.

Example 1

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

```
const puppeteer = require('puppeteer');

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

The Page class extends from Puppeteer's [EventEmitter](#) class and will emit various events which are documented in the [PageEmittedEvents](#) enum.

Example 2

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

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

To unsubscribe from events use the `off` method:

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

Properties

Property	Modifiers	Type	Description
accessibility		Accessibility	
coverage		Coverage	
keyboard		Keyboard	
mouse		Mouse	
touchscreen		Touchscreen	
tracing		Tracing	

Methods

Method	Modifiers	Description
\$(selector)		Runs <code>document.querySelector</code> within the page. If an element matches the selector, the return value resolves to <code>null</code> .
\$\$ (selector)		The method runs <code>document.querySelectorAll</code> within the page. If no elements match the selector, the return value resolves to <code>[]</code> .
\$\$eval(selector, pageFunction, args)		This method runs <code>Array.from(document.querySelectorAll(selector), pageFunction, ...args)</code> within the page and passes the result as the first argument to the <code>pageFunction</code> .
\$eval(selector, pageFunction, args)		This method runs <code>document.querySelector(selector).pageFunction(...args)</code> within the page and passes the result as the first argument to the <code>pageFunction</code> .
\$x(expression)		The method evaluates the XPath expression relative to the page document as its context node. If there are no such elements, the method resolves to an empty array.
addScriptTag(options)		Adds a <code><script></code> tag into the page with the desired URL or content.
addStyleTag(options)		Adds a <code><link rel="stylesheet"></code> tag into the page with the desired URL or a <code><style type="text/css"></code> tag with the content.
authenticate(credentials)		Provide credentials for HTTP authentication.
bringToFront()		Brings page to front (activates tab).
browser()		Get the browser the page belongs to.

browserContext()		Get the browser context that the page belongs to.
click(selector, options)		This method fetches an element with <code>selector</code> , scrolls it into view if needed, and then uses Page.mouse to click in the center of the element. If there's no element matching <code>selector</code> , the method throws an error.
close(options)		
content()		
cookies(urls)		If no URLs are specified, this method returns cookies for the current page URL. If URLs are specified, only cookies for those URLs are returned.
createPDFStream(options)		Generates a PDF of the page with the <code>print</code> CSS media type.
deleteCookie(cookies)		
emulate(options)		Emulates given device metrics and user agent. This method is a shortcut for calling two methods: Page.setUserAgent() and Page.setViewport() . To emulate, Puppeteer provides a list of device descriptors that can be obtained via the Puppeteer.devices page. <code>page.emulate</code> will resize the page. A lot of websites don't expect phones to change size, so you should emulate before navigating to the page.
emulateCPUThrottling(factor)		
emulateIdleState(overrides)		Emulates the idle state. If no arguments set, clears idle state emulation.
emulateMediaFeatures(features)		
emulateMediaType(type)		
emulateNetworkConditions(networkConditions)		
emulateTimezone(timezoneId)		
emulateVisionDeficiency(type)		Simulates the given vision deficiency on the page.
evaluate(pageFunction, args)		
evaluateHandle(pageFunction, args)		
evaluateOnNewDocument(pageFunction, args)		Adds a function which would be invoked in one of the following scenarios:- whenever the page is navigated- whenever the child frame is attached and navigated. In this case, the function is invoked in the context of the newly attached frame. The function is invoked after the document was created.

		but before any of its scripts were run. This is used to amend the JavaScript environment, e.g. to set <code>Math.random</code> .
<code>exposeFunction(name, puppeteerFunction)</code>		The method adds a function called <code>name</code> on the page's <code>window</code> object. When called, the function executes <code>puppeteerFunction</code> in node.js and returns a <code>Promise</code> which resolves to the return value of <code>puppeteerFunction</code> . If the <code>puppeteerFunction</code> returns a <code>Promise</code> , it will be awaited. NOTE: Functions installed via <code>page.exposeFunction</code> survive navigations.
<code>focus(selector)</code>		This method fetches an element with <code>selector</code> and focuses it. If there's no element matching <code>selector</code> , the method throws an error.
<code>frames()</code>		
<code>goBack(options)</code>		This method navigates to the previous page in history.
<code>goForward(options)</code>		This method navigates to the next page in history.
<code>goto(url, options)</code>		
<code>hover(selector)</code>		This method fetches an element with <code>selector</code> , scrolls it into view if needed, and then uses <code>Page.mouse</code> to hover over the center of the element. If there's no element matching <code>selector</code> , the method throws an error.
<code>isClosed()</code>		Indicates that the page has been closed.
<code>isDragInterceptionEnabled()</code>		
<code>isJavaScriptEnabled()</code>		
<code>mainFrame()</code>		
<code>metrics()</code>		
<code>on(eventName, handler)</code>		Listen to page events.
<code>once(eventName, handler)</code>		
<code>pdf(options)</code>		
<code>queryObjects(prototypeHandle)</code>		This method iterates the JavaScript heap and finds all objects with the given prototype.
<code>reload(options)</code>		
<code>screenshot(options)</code>		
<code>select(selector, values)</code>		Triggers a <code>change</code> and <code>input</code> event once all the provided options have been selected. If there's

		<code><selector></code> element matching <code>selector</code> , the method throws an error.
<code>setBypassCSP(enabled)</code>		Toggles bypassing page's Content-Security-Policy.
<code>setCacheEnabled(enabled)</code>		Toggles ignoring cache for each request based on the enabled state. By default, caching is enabled.
<code>setContent(html, options)</code>		
<code>setCookie(cookies)</code>		
<code>setDefaultNavigationTimeout(timeout)</code>		This setting will change the default maximum navigation time for the following methods and related shortcuts:- <code>page.goBack(options)</code> ,- <code>page.goForward(options)</code> ,- <code>page.goto(url,options)</code> ,- <code>page.reload(options)</code> ,- <code>page.setContent(html,options)</code> ,- <code>page.waitForNavigation(options)</code> .
<code>setDefaultTimeout(timeout)</code>		
<code>setDragInterception(enabled)</code>		
<code>setExtraHTTPHeaders(headers)</code>		The extra HTTP headers will be sent with every request the page initiates. NOTE: All HTTP header names are lowercased. (HTTP headers are case-insensitive, so this shouldn't impact your server code.) NOTE: <code>page.setExtraHTTPHeaders</code> does not guarantee the order of headers in the outgoing requests.
<code>setGeolocation(options)</code>		Sets the page's geolocation.
<code>setJavaScriptEnabled(enabled)</code>		
<code>setOfflineMode(enabled)</code>		
<code>setRequestInterception(value)</code>		
<code>setUserAgent(userAgent, userAgentMetadata)</code>		
<code>setViewport(viewport)</code>		<code>page.setViewport</code> will resize the page. A lot of websites don't expect phones to change size, so you should set the viewport before navigating to the page. In the case of multiple pages in a single browser, each page can have its own viewport settings.
<code>tap(selector)</code>		This method fetches an element with <code>selector</code> , scrolls it into view if needed, and then uses <code>Page.touchscreen</code> to tap in the center of the element. If there's no element matching <code>selector</code> , the method throws an error.
<code>target()</code>		

title()		
type(selector, text, options)		Sends a <code>keydown</code> , <code>keypress/input</code> , and <code>keyup</code> event for each character in the text. To press a special key like <code>Control</code> or <code>ArrowDown</code> , use Keyboard.press() .
url()		
viewport()		
waitFor(selectorOrFunctionOrTimeout, options, args)		
waitForFileChooser(options)		This method is typically coupled with an action that triggers file choosing. The following example clicks a button that issues a file chooser and then responds with <code>/tmp/myfile.pdf</code> as if a user has selected this file.

```
const [fileChooser] = await Promise.all([
  page.waitForFileChooser(),
  page.click('#upload-file-button'),
  // some button that triggers file selection
]);
await fileChooser.accept(['/tmp/myfile.pdf']);
```

NOTE: This must be called before the file chooser is launched. It will not return a currently active file chooser. || [waitForFunction\(pageFunction, options, args\)](#) || The `waitForFunction` can be used to observe viewport size change:

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

To pass arguments from node.js to the predicate of `page.waitForFunction` function:

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

The predicate of `page.waitForFunction` can be asynchronous too:

```

const username = 'github-username';
await page.waitForFunction(
  async (username) => {
    const githubResponse = await fetch(
      `https://api.github.com/users/${username}`
    );
    const githubUser = await githubResponse.json();
    // show the avatar
    const img = document.createElement('img');
    img.src = githubUser.avatar_url;
    // wait 3 seconds
    await new Promise((resolve, reject) => setTimeout(resolve, 3000));
    img.remove();
  },
  {},
  username
);

```

|| [waitForNavigation\(options\)](#) || This resolves when the page navigates to a new URL or reloads. It is useful when you run code that will indirectly cause the page to navigate. Consider this example:

```

const [response] = await Promise.all([
  page.waitForNavigation(), // The promise resolves after navigation has finished
  page.click('a.my-link'), // Clicking the link will indirectly cause a navigation
]);

```

|| [waitForRequest\(urlOrPredicate, options\)](#) || || [waitForResponse\(urlOrPredicate, options\)](#) || || ||

[waitForSelector\(selector, options\)](#) || Wait for the `selector` to appear in page. If at the moment of calling the method the `selector` already exists, the method will return immediately. If the `selector` doesn't appear after the `timeout` milliseconds of waiting, the function will throw. This method works across navigations:

```

const puppeteer = require('puppeteer');
(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  let currentURL;
  page
    .waitForSelector('img')
    .then(() => console.log('First URL with image: ' + currentURL));
  for (currentURL of [
    'https://example.com',
    'https://google.com',
    'https://bbc.com',
  ]) {
    await page.goto(currentURL);
  }
  await browser.close();
})();

```

[|| waitForTimeout\(milliseconds\) ||](#) Causes your script to wait for the given number of milliseconds. ||

[waitForXPath\(xpath, options\) ||](#) Wait for the `xpath` to appear in page. If at the moment of calling the method the `xpath` already exists, the method will return immediately. If the `xpath` doesn't appear after the `timeout` milliseconds of waiting, the function will throw. This method works across navigation

```
const puppeteer = require('puppeteer');
(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  let currentURL;
  page
    .waitForXPath('//img')
    .then(() => console.log('First URL with image: ' + currentURL));
  for (currentURL of [
    'https://example.com',
    'https://google.com',
    'https://bbc.com',
  ]) {
    await page.goto(currentURL);
  }
  await browser.close();
})();
```

[|| workers\(\) ||](#)