Skip to main content

PlaywrightDocsAPI
Node.js

- Node.js
- Python
- Java
- .NET

Community
Search⌘K

Pages

Pages

Each BrowserContext can have multiple pages. A Page refers to a single tab or a popup window within a browser context. It should be used to navigate to URLs and interact with the page content.

```
// Create a page.
const page = await context.newPage();

// Navigate explicitly, similar to entering a URL in the browser.
await page.goto('http://example.com');
// Fill an input.
await page.locator('#search').fill('query');

// Navigate implicitly by clicking a link.
await page.locator('#submit').click();
// Expect a new url.
console.log(page.url());
```

Multiple pages

Each browser context can host multiple pages (tabs).

- Each page behaves like a focused, active page. Bringing the page to front is not required.
- Pages inside a context respect context-level emulation, like viewport sizes, custom network routes or browser locale.

```
// Create two pages
const pageOne = await context.newPage();
const pageTwo = await context.newPage();

// Get pages of a browser context
const allPages = context.pages();
```

Handling new pages

The `page` event on browser contexts can be used to get new pages that are created in the context. This can be used to handle new pages opened by `target="_blank"` links.

```
// Start waiting for new page before clicking. Note no await.
const pagePromise = context.waitForEvent('page');
await page.getByText('open new tab').click();
const newPage = await pagePromise;
// Interact with the new page normally.
await newPage.getByRole('button').click();
console.log(await newPage.title());
```

If the action that triggers the new page is unknown, the following pattern can be used.

```
// Get all new pages (including popups) in the context
context.on('page', async page => {
  await page.waitForLoadState();
  console.log(await page.title());
});
```

Handling popups

If the page opens a pop-up (e.g. pages opened by `target="_blank"` links), you can get a reference to it by listening to the `popup` event on the page.

This event is emitted in addition to the `browserContext.on('page')` event, but only for popups relevant to this page.

```
// Start waiting for popup before clicking. Note no await.
const popupPromise = page.waitForEvent('popup');
await page.getByText('open the popup').click();
const popup = await popupPromise;
// Interact with the new popup normally.
await popup.getByRole('button').click();
console.log(await popup.title());
```

If the action that triggers the popup is unknown, the following pattern can be used.

```
// Get all popups when they open
page.on('popup', async popup => {
  await popup.waitForLoadState();
  console.log(await popup.title());
});
```

Previous
Page object models

Next
Screenshots

Learn