

[Skip to main content](#)



[Playwright Docs API](#)

[Node.js](#)

- [Node.js](#)
- [Python](#)
- [Java](#)
- [.NET](#)

[Community](#)

Search

- [Getting Started](#)
 - [Installation](#)
 - [Writing tests](#)
 - [Generating tests](#)
 - [Running and debugging tests](#)
 - [Trace viewer](#)
 - [Setting up CI](#)
- [Getting started - VS Code](#)
- [Release notes](#)
- [Canary releases](#)
- [Playwright Test](#)
 - [Test configuration](#)
 - [Test use options](#)
 - [Annotations](#)
 - [Command line](#)

- [Emulation](#)
- [Fixtures](#)
- [Global setup and teardown](#)
- [Parallelism](#)
- [Parameterize tests](#)
- [Projects](#)
- [Reporters](#)
- [Retries](#)
- [Sharding](#)
- [Timeouts](#)
- [TypeScript](#)
- [UI Mode](#)
- [Web server](#)
- [Guides](#)
 - [Library](#)
 - [Accessibility testing](#)
 - [Actions](#)
 - [Assertions](#)
 - [API testing](#)
 - [Authentication](#)
 - [Auto-waiting](#)
 - [Best Practices](#)
 - [Browsers](#)
 - [Chrome extensions](#)
 - [Clock](#)
 - [Components \(experimental\)](#)
 - [Debugging Tests](#)
 - [Dialogs](#)
 - [Downloads](#)
 - [Evaluating JavaScript](#)
 - [Events](#)
 - [Extensibility](#)
 - [Frames](#)
 - [Handles](#)
 - [Isolation](#)
 - [Locators](#)
 - [Mock APIs](#)
 - [Mock browser APIs](#)
 - [Navigations](#)
 - [Network](#)
 - [Other locators](#)
 - [Page object models](#)
 - [Pages](#)
 - [Screenshots](#)
 - [Visual comparisons](#)
 - [Test generator](#)
 - [Trace viewer](#)
 - [Videos](#)
 - [WebView2](#)
- [Migration](#)

- [Integrations](#)
- [Supported languages](#)
-
- Guides
- Debugging Tests

On this page

Debugging Tests

VS Code debugger

We recommend using the [VS Code Extension](#) for debugging for a better developer experience. With the VS Code extension you can debug your tests right in VS Code, see error messages, set breakpoints and step through your tests.

A screenshot of a developer tools interface, likely from a browser or a dedicated testing tool. The interface has a dark theme with orange and white highlights.

Top Bar:

- Three circular icons (red, yellow, green) in the top-left corner.
- Text: "RU..." followed by a play button icon and "No Conf ▾".
- A gear icon and three dots (...).
- A test tube icon labeled "exam" and a "tests > [number]" counter.

Sidebar (Left):

- Icons: file, magnifying glass, network, eye (with a '1' badge), ellipsis, user, and gear.

Main Content Area:

- VARIABLES**:
 - Local**:
 - > `page: Page` {`_events: {...}`...}
 - `this: undefined`
 - > **Closure**
 - > **Global**
- BREAKPOINTS**:
 - Caught Exceptions
 - Uncaught Exceptions
 - example.s... (13) (13)
- WATCH**: (empty)

Bottom Bar:

- Icons: error (0), warning (0), and a running status indicator.
- Text: "Running tests..."

Test Counter on the Right:

- Numbers 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18.
- Icons: a hexagon with a dot, a yellow square, and a green curly brace.

Error Messages

If your test fails VS Code will show you error messages right in the editor showing what was expected, what was received as well as a complete call log.

The screenshot shows the Playwright extension in the Visual Studio Code interface. The sidebar on the left has icons for file, search, connections, and Playwright. The main area is titled "TESTING" and contains a "TEST EXPLORER". A filter bar says "Filter (e.g. text, !exclu...)" with a dropdown icon. It shows "0/1 tests passed (0.00%)". Under "tests" (with a red error icon), there are three items: "example.spec.ts" (1.0s), "has title" (866ms) with a green checkmark, and "get started link" (150ms) with a red error icon. Below the test explorer is a "PLAYWRIGHT" section with status icons for errors (0) and warnings (0). To the right, a preview pane shows code snippets for locator and get started.

TESTING

TEST EXPLORER

Filter (e.g. text, !exclu...)

0/1 tests passed (0.00%)

- ✗ tests 1.0s
 - ✗ example.spec.ts 1.0s
 - ✓ has title 866ms
 - ✗ get started link 150ms

PLAYWRIGHT

✗ 0 ⚠ 0

locator.c

locator.c

1) <a href="#" data-

2) <a href="#" data-

btn" target="

 aka

3) <a href="#" data-

count" re

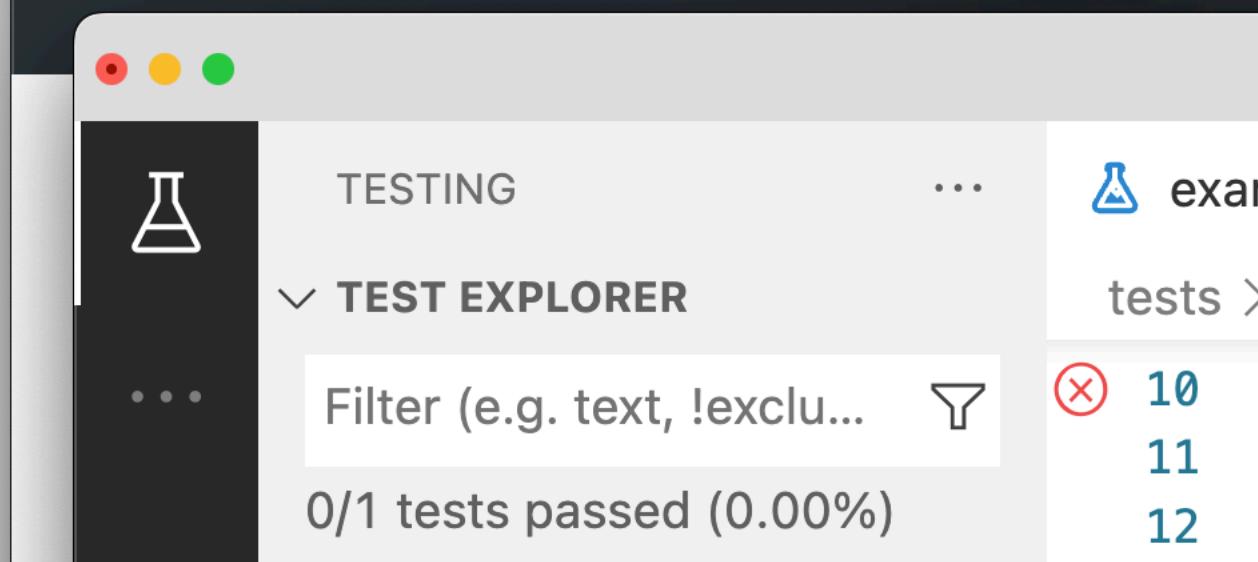
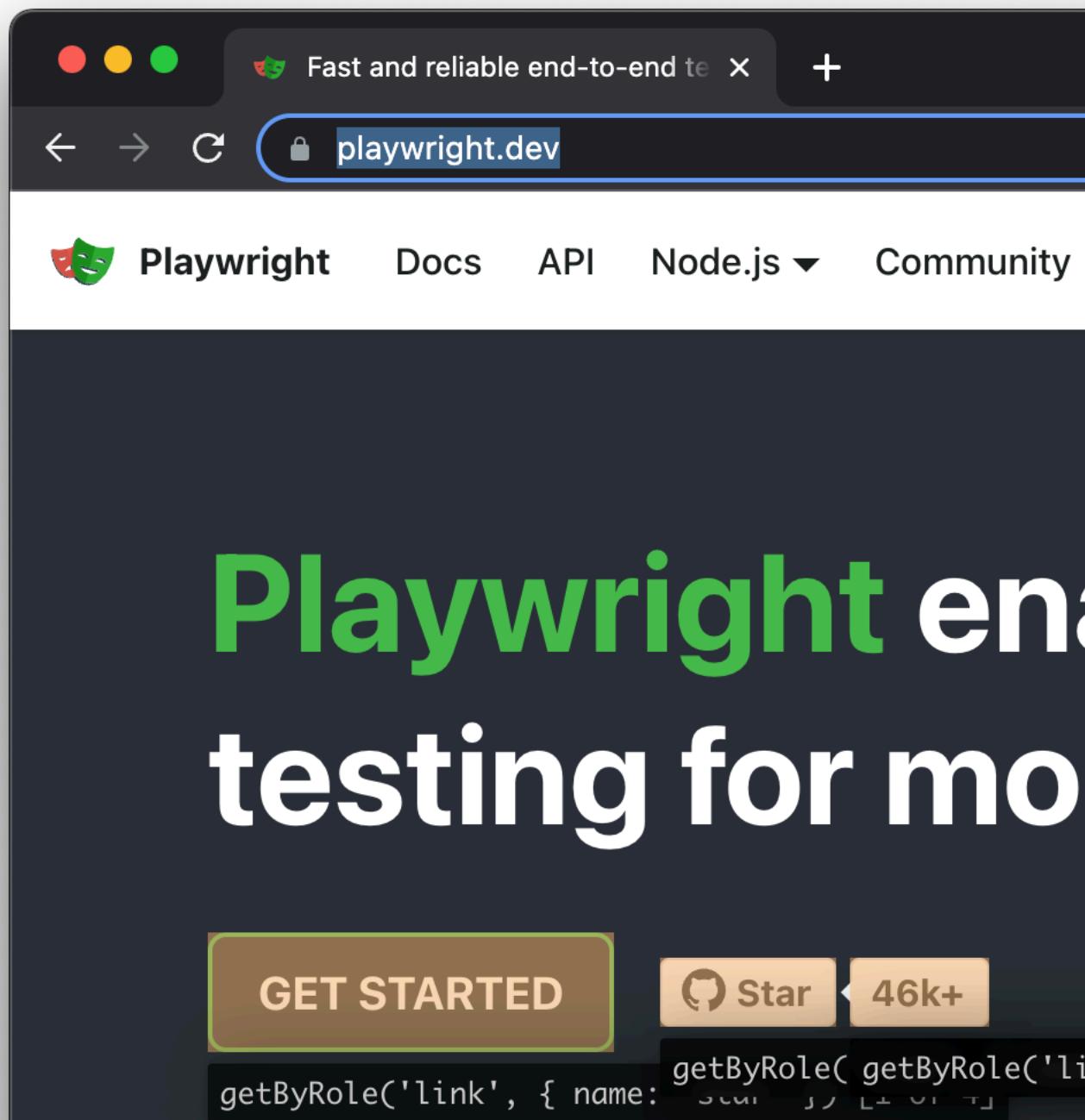
 aka

4) <a href="#" data-

item">G

Live Debugging

You can debug your test live in VS Code. After running a test with the `Show Browser` option checked, click on any of the locators in VS Code and it will be highlighted in the Browser window. Playwright will also show you if there are multiple matches.



You can also edit the locators in VS Code and Playwright will show you the changes live in the browser window.

A screenshot of a web browser window displaying the Playwright homepage at `playwright.dev`. The browser interface includes standard controls like back, forward, and search. The page features a large green and white title: "Playwright enables testing for more". Below the title is a prominent orange "GET STARTED" button. To its right are a "Star" icon and a "46k+" badge. A code snippet, `getByRole('link', { name: 'started' }) [1 of 2]`, is shown in a dark box. The overall theme is dark with light-colored text and highlights.

Fast and reliable end-to-end te X +

← → C 🔒 playwright.dev

Playwright Docs API Node.js ▾ Community

Playwright enables testing for more

GET STARTED

Star 46k+

```
getByRole('link', { name: 'started' }) [1 of 2]
```

A screenshot of a dark-themed IDE or developer tool interface. On the left is a sidebar with a flask icon and a "TESTING" section. The main area is titled "TEST EXPLORER" and shows a "Filter (e.g. text, !excl...)" input field. At the bottom, it displays "0/1 tests passed (0.00%)". To the right, there are several numbered items: "10", "11", and "12", each with a small red circle containing a white "X". The overall aesthetic is modern and minimalist.

TESTING

TEST EXPLORER

Filter (e.g. text, !excl...)

0/1 tests passed (0.00%)

10 11 12

Picking a Locator

Pick a [locator](#) and copy it into your test file by clicking the **Pick locator** button from the testing sidebar. Then in the browser click the element you require and it will now show up in the **Pick locator** box in VS Code. Press 'enter' on your keyboard to copy the locator into the clipboard and then paste anywhere in your code. Or press 'escape' if you want to cancel.

The screenshot shows a web browser window with the title "Installation | Playwright". The URL in the address bar is "playwright.dev/docs/intro". The page content is the "Getting Started" section of the documentation, specifically the "Installation" subsection. The sidebar on the left lists various documentation topics. A large, semi-transparent modal window is overlaid on the page, containing a dark interface with icons and text related to testing and Playwright features.

Playwright Docs API Node.js ▾ Community

Getting Started

Installation

Writing Tests

Running Tests

Test Generator

Trace Viewer

CI GitHub Actions

Getting started - VS Code

Release notes

Canary Releases

Playwright Test

Annotations

API testing

Assertions

Command line

Configuration

TESTING

TEST EXPLORE

Filter (e.g. text)

1/1 tests passed (

tests 1.1s

example.sp

has title

get starte

PLAYWRIGHT

Show browser

Pick locator

Record new

Record at cursor

Reveal test output

Playwright will look at your page and figure out the best locator, prioritizing [role, text and test id locators](#). If Playwright finds multiple elements matching the locator, it will improve the locator to make it resilient and uniquely identify the target element, so you don't have to worry about failing tests due to locators.

Run in Debug Mode

To set a breakpoint click next to the line number where you want the breakpoint to be until a red dot appears. Run the tests in debug mode by right clicking on the line next to the test you want to run.

The screenshot shows the VS Code interface with the Test Explorer extension open. The sidebar on the left contains icons for file, search, connections, and test (with a '1' notification). The main area is titled "TESTING" and shows the "TEST EXPLORER". A filter bar allows for filtering tests (e.g., text, !exclude). The status message indicates "0/1 tests passed (0.00%)". A tree view displays a single test: "tests 1.0s" which includes "example.spec.ts 1.0s". Under "example.spec.ts", two tests are listed: "has title" (passed, green checkmark) and "get started link" (failed, red X). The "PLAYWRIGHT" section at the bottom shows 0 errors and 0 warnings. The right side of the screen shows a vertical list of numbered items from 4 to 18, with some entries partially cut off.

TESTING

...

TEST EXPLORER

Filter (e.g. text, !exclu...)

0/1 tests passed (0.00%)

✗ tests 1.0s

 ✗ example.spec.ts 1.0s

 ✓ has title 866ms

 ✗ get started link 150ms

...

PLAYWRIGHT

✖ 0 ⚠ 0

4

5

6

7

8

9

✗ 10

11

12

● 13

14

15

16

17

18

A browser window will open and the test will run and pause at where the breakpoint is set. You can step through the tests, pause the test and rerun the tests from the menu in VS Code.

A screenshot of a developer tools interface, likely from a browser or a dedicated testing tool. The interface has a dark theme with orange and white highlights.

Top Bar:

- Three circular icons (red, yellow, green) in the top-left corner.
- Text: "RU..."
- Icon: Green play button with a triangle.
- Text: "No Conf" with a dropdown arrow.
- Icon: Gear.
- Text: "..."

Left Sidebar:

- Icon: File.
- Icon: Magnifying glass.
- Icon: Network.
- Breakpoint icon:** A blue circle with a white number "1".
- Text: "..."
- Icon: User.
- Icon: Gear.

Main Content Area:

- VARIABLES**
 - Local**
 - > **page:** Page { _events: { ... }... }
 - > **this:** undefined
 - > **Closure**
 - > **Global**
- BREAKPOINTS**
 - Caught Exceptions
 - Uncaught Exceptions
 - example.s... (13) (13)
- WATCH**

Bottom Status Bar:

- Icon: Circle with X: 0
- Icon: Triangle with exclamation mark: 0
- Icon: Refresh: Running tests...

Debug in different Browsers

By default, debugging is done using the Chromium profile. You can debug your tests on different browsers by right clicking on the debug icon in the testing sidebar and clicking on the 'Select Default Profile' option from the dropdown.

TESTING

...

example.spec.ts tests > chromium — docs/playwright.config.js

firefox — docs/playwright.config.ts

webkit — docs/playwright.config.ts

Filter (e.g. text, !)

0/1 tests passed

✗ tests 727ms

✗ example.spec.ts 727ms

○ has title

✗ get started link 727ms

Select Default Profile

5

6

7

8

9

10

11

12

13

14

15

16

17

18

✗ PLAYWRIGHT

Show browser

🔗 Pick locator

➕ Record new

✍ Record at cursor

✉ Reveal test output

✖ Close all browsers

Then choose the test profile you would like to use for debugging your tests. Each time you run your test in debug mode it will use the profile you selected. You can run tests in debug mode by right clicking the line number where your test is and selecting 'Debug Test' from the menu.

The screenshot shows a dark-themed user interface for a testing tool. On the left is a vertical sidebar with icons for file operations, search, network, media, and windows. The main area has a title "TESTING" and a dropdown menu showing "chromium — docs/playwright" is selected. A "Filter (e.g. text,)" input field is present. The test results show "0/1 tests passed". A tree view lists a single test file "example.spec.ts" with one failing test: "get started link". The failing test is highlighted with a red circle and a red cross icon. The right side of the interface displays a numbered list of steps or logs, with some entries having red or green status indicators.

Pick a test profile to use

chromium — docs/playwright

firefox — docs/playwright

webkit — docs/playwright

tests 727ms

example.spec.ts 727ms

has title

get started link 727ms

PLAYWRIGHT

Show browser

Pick locator

Record new

Record at cursor

Reveal test output

Close all browsers

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

To learn more about debugging, see [Debugging in Visual Studio Code](#).

Playwright Inspector

The Playwright Inspector is a GUI tool to help you debug your Playwright tests. It allows you to step through your tests, live edit locators, pick locators and see actionability logs.

Playwright Inspector

Record ⌂ ▶ || ⌂ Tar

```
1 import { test, expect } from '@playwright/test';
2
3 test('has title', async ({ page }) => {
4   await page.goto('https://playwright.dev');
5
6   // Expect a title "to contain" a substring
7   await expect(page).toHaveTitle(/Playwright/);
8 });
9
10 test('get started link', async ({ page }) => {
11   await page.goto('https://playwright.dev');
12   // Click the get started link.
13   await page.getByRole('link', { name: 'Get started' });
14
15   // Expects the URL to contain intro.
16   await expect(page).toHaveURL(/.*intro/);
17 });

A Pick locator    getByRole('link', { name: 'Get started' })

> browserContext.newPage ✓ — 222ms
> page.goto('https://playwright.dev') ✓ — 405ms
▼ page.getByRole('link', { name: 'Get started' }).click() ||
  waiting for getByRole('link', { name: 'Get started' })
  locator resolved to visible <a href="/docs/intro" class="nav-link">
  attempting click action
  waiting for element to be visible, enabled and stable
  element is visible, enabled and stable
```

Run in debug mode

Run your tests with the `--debug` flag to open the inspector. This configures Playwright for debugging and opens the inspector. Additional useful defaults are configured when `--debug` is used:

- Browsers launch in headed mode
- Default timeout is set to 0 (= no timeout)

Debug all tests on all browsers

To debug all tests run the test command with the `--debug` flag. This will run tests one by one, and open the inspector and a browser window for each test.

```
npx playwright test --debug
```

Debug one test on all browsers

To debug one test on a specific line, run the test command followed by the name of the test file and the line number of the test you want to debug, followed by the `--debug` flag. This will run a single test in each browser configured in your [playwright.config](#) and open the inspector.

```
npx playwright test example.spec.ts:10 --debug
```

Debug on a specific browser

In Playwright you can configure projects in your [playwright.config](#). Once configured you can then debug your tests on a specific browser or mobile viewport using the `--project` flag followed by the name of the project configured in your `playwright.config`.

```
npx playwright test --project=chromium --debug  
npx playwright test --project="Mobile Safari" --debug  
npx playwright test --project="Microsoft Edge" --debug
```

Debug one test on a specific browser

To run one test on a specific browser add the name of the test file and the line number of the test you want to debug as well as the `--project` flag followed by the name of the project.

```
npx playwright test example.spec.ts:10 --project=webkit --debug
```

Stepping through your tests

You can play, pause or step through each action of your test using the toolbar at the top of the Inspector. You can see the current action highlighted in the test code, and matching elements highlighted in the browser window.

A screenshot of a web browser displaying the official Playwright website. The browser interface includes a top bar with three dots, a title bar showing the Playwright logo and the text "Fast and reliable end-to-end te", a tab labeled "+", and a URL bar with a lock icon and the address "playwright.dev". Below the browser window, the Playwright navigation bar features the logo, "Playwright", "Docs", "API", "Node.js", and "Community". The main content area has a dark background with large, bold text: "Playwright enables fast and reliable end-to-end testing for modern web applications". A prominent blue button with white text says "GET STARTED". To its right are a GitHub star icon and the text "46k+". Below this is a code snippet: `getByRole('link', { name: 'Get started' })`. In the bottom right corner of the browser window, there is a small circular icon containing the Google Chrome logo.

Fast and reliable end-to-end te

playwright.dev

Playwright Docs API Node.js Community

Playwright enables fast and reliable end-to-end testing for modern web applications

[GET STARTED](#)

Star 46k+

```
getByRole('link', { name: 'Get started' })
```

Run a test from a specific breakpoint

To speed up the debugging process you can add a [page.pause\(\)](#) method to your test. This way you won't have to step through each action of your test to get to the point where you want to debug.

```
await page.pause();
```

Once you add a `page.pause()` call, run your tests in debug mode. Clicking the "Resume" button in the Inspector will run the test and only stop on the `page.pause()`.

A screenshot of a mobile web browser displaying the official Playwright website at `playwright.dev`. The browser's header shows the title "Fast and reliable end-to-end te" and the address bar shows the URL. The website features a dark background with large, bold text: "Playwright en" in green and "testing for mo" in white. Below this is a green-outlined button labeled "GET STARTED". To the right of the button is a GitHub star icon followed by "47k+". A small Chrome icon is visible in the bottom right corner of the page area.

Fast and reliable end-to-end te

playwright.dev

Playwright Docs API Node.js ▾ Community

Playwright en

testing for mo

GET STARTED

Star 47k+



Live editing locators

While running in debug mode you can live edit the locators. Next to the 'Pick Locator' button there is a field showing the [locator](#) that the test is paused on. You can edit this locator directly in the **Pick Locator** field, and matching elements will be highlighted in the browser window.

The screenshot shows a mobile browser displaying the [Playwright documentation](https://playwright.dev/docs/intro) at the URL `https://playwright.dev/docs/intro`. The page is titled "Getting Started" and features a sidebar with navigation links. The "Installation" link is highlighted, indicating it is the current section being viewed.

Playwright Documentation

Getting Started

- Installation
- Writing Tests
- Running Tests
- Test Generator
- Trace Viewer
- CI GitHub Actions
- Getting started - VS Code
- Release notes
- Canary Releases

Playwright Test

- Annotations
- API testing
- Assertions
- Command line
- Configuration
- Parallelism and sharding

Installation

getByRole('heading', {
Playwright Test was created
Playwright supports all
on Windows, Linux, and
emulation of Google Ch

You will learn

- How to install Playwright
- What's Installed
- How to run the examples
- How to open the Help

Installing Playwright

getByRole('heading', {
Get started by installing
run your tests using the

Picking locators

While debugging, you might need to choose a more resilient locator. You can do this by clicking on the **Pick Locator** button and hovering over any element in the browser window. While hovering over an element you will see the code needed to locate this element highlighted below. Clicking an element in the browser will add the locator into the field where you can then either tweak it or copy it into your code.

A screenshot of a web browser displaying the Playwright documentation at <https://playwright.dev/docs/intro>. The page is titled "Getting Started" and features a sidebar with links to "Installation", "Writing Tests", "Running Tests", "Test Generator", "Trace Viewer", "CI GitHub Actions", "Getting started - VS Code", "Release notes", and "Canary Releases". Below this is a section titled "Playwright Test" with links to "Annotations", "API testing", "Assertions", "Command line", "Configuration", and "Parallelism and sharding". To the right, a main content area shows the "Installation" section, which includes a heading "Install", a snippet of code using `getByRole('heading')`, and a "You will learn" section with four bullet points: "How to install", "What's Installed", "How to run the", and "How to open t". Further down, there is a large "Installing" heading and a paragraph about getting started with installation.

Playwright Docs API Node.js ▾ Community

Getting Started

Installation

Writing Tests

Running Tests

Test Generator

Trace Viewer

CI GitHub Actions

Getting started - VS Code

Release notes

Canary Releases

Playwright Test

Annotations

API testing

Assertions

Command line

Configuration

Parallelism and sharding

Installation

```
getByRole('heading')
```

Playwright Test was c
Playwright supports a
on Windows, Linux, a
emulation of Google

You will learn

- How to install
- What's Installed
- How to run the
- How to open t

Installing

Get started by inst
run your tests usin

Playwright will look at your page and figure out the best locator, prioritizing [role, text and test id locators](#). If Playwright finds multiple elements matching the locator, it will improve the locator to make it resilient and uniquely identify the target element, so you don't have to worry about failing tests due to locators.

Actionability logs

By the time Playwright has paused on a click action, it has already performed [actionability checks](#) that can be found in the log. This can help you understand what happened during your test and what Playwright did or tried to do. The log tells you if the element was visible, enabled and stable, if the locator resolved to an element, scrolled into view, and so much more. If actionability can't be reached, it will show the action as pending.

The screenshot shows the Playwright Inspector interface. At the top, there are three red, yellow, and green window control buttons. To the right of them is the title "Playwright Inspector". Below the title is a toolbar with several icons: a circular "Record" button, a clipboard icon, a play/pause button, a double vertical bar button, and a refresh/circular arrow button.

The main area contains a code editor window displaying a portion of a JavaScript test script:

```
10 test('get started link', async ({ page }) {
11   await page.goto('https://playwright.dev')
12   // Click the get started link.
13   await page.getByRole('link', { name: 'Get started' })
14
15   // Expects the URL to contain intro.
16   await expect(page).toHaveURL(/.*intro/)
17});
```

Below the code editor is a section titled "Pick locator" with a dropdown menu containing the text `locator(':root')`.

The bottom half of the screen displays a timeline of the test execution steps:

- > `browserContext.newPage` ✓ — 293ms
- ✓ `page.goto('https://playwright.dev')` ✓ — 510ms
 - navigating to "https://playwright.dev/", waiting until "load"
- ✓ `page.getByRole('link', { name: 'Get started' }).click()` ✓ —
 - waiting for `getByRole('link', { name: 'Get started' })`
 - locator resolved to visible
 - attempting click action
 - waiting for element to be visible, enabled and stable
 - element is visible, enabled and stable
 - scrolling into view if needed
 - done scrolling
 - performing click action
 - click action done
 - waiting for scheduled navigations to finish

Trace Viewer

Playwright [Trace Viewer](#) is a GUI tool that lets you explore recorded Playwright traces of your tests. You can go back and forward through each action on the left side, and visually see what was happening during the action. In the middle of the screen, you can see a DOM snapshot for the action. On the right side you can see action details, such as time, parameters, return value and log. You can also explore console messages, network requests and the source code.

Your browser does not support the video tag.

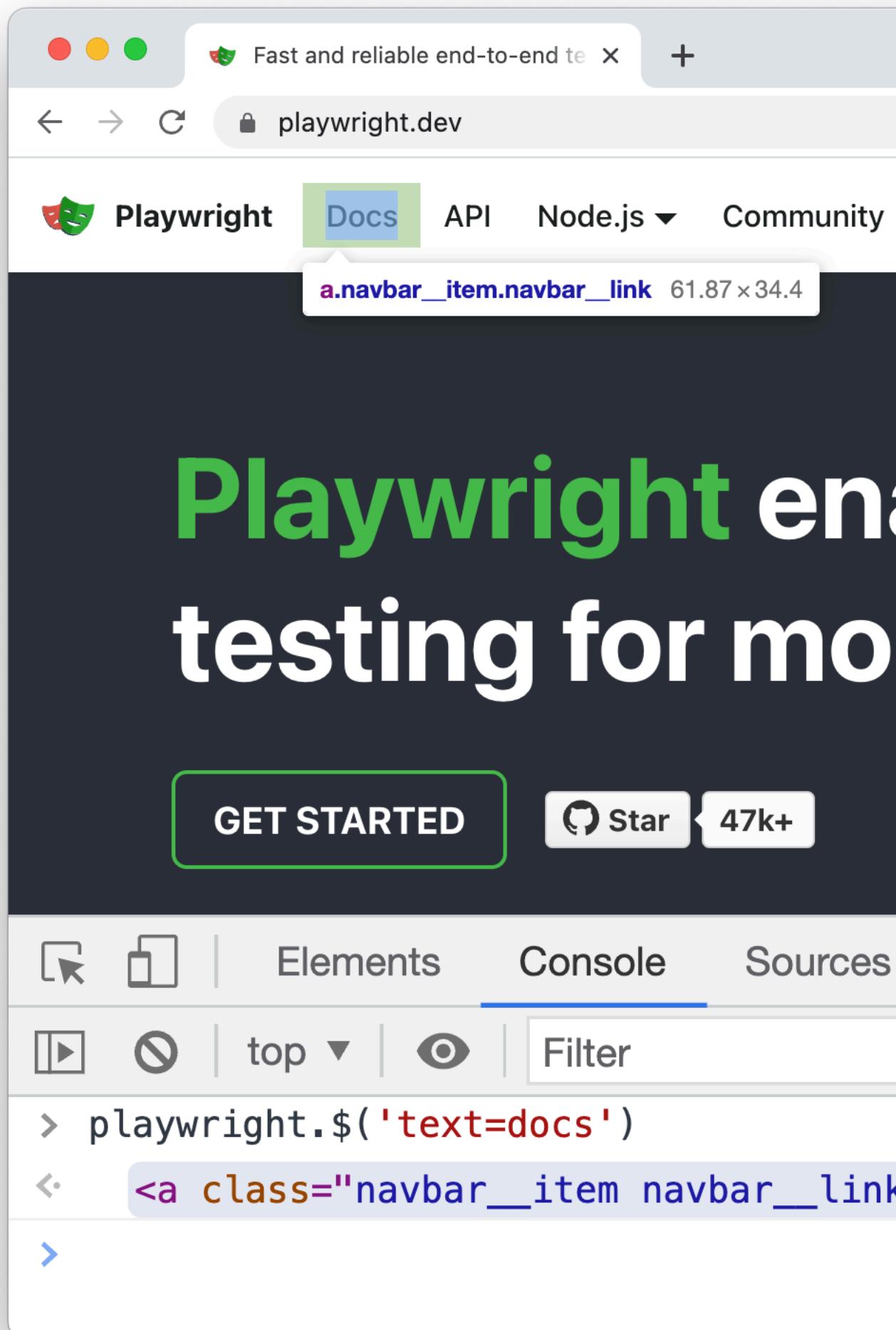
To learn more about how to record traces and use the Trace Viewer, check out the [Trace Viewer](#) guide.

Browser Developer Tools

When running in Debug Mode with `PWDEBUG=console`, a `playwright` object is available in the Developer tools console. Developer tools can help you to:

- Inspect the DOM tree and **find element selectors**
- See **console logs** during execution (or learn how to [read logs via API](#))
- Check **network activity** and other developer tools features

This will also set the default timeouts of Playwright to 0 (= no timeout).



To debug your tests using the browser developer tools, start by setting a breakpoint in your test to pause the execution using the [page.pause\(\)](#) method.

```
await page.pause();
```

Once you have set a breakpoint in your test, you can then run your test with `PWDEBUG=console`.

- Bash
- PowerShell
- Batch

```
PWDEBUG=console npx playwright test
$env:PWDEBUG="console"
npx playwright test
set PWDEBUG=console
npx playwright test
```

Once Playwright launches the browser window, you can open the developer tools. The `playwright` object will be available in the console panel.

playwright.\$(selector)

Query the Playwright selector, using the actual Playwright query engine, for example:

```
playwright.$('.auth-form >> text=Log in');
<button>Log in</button>
```

playwright.\$\$(selector)

Same as `playwright.$`, but returns all matching elements.

```
playwright.$$('li >> text=John')
[<li>, <li>, <li>, <li>]
```

playwright.inspect(selector)

Reveal element in the Elements panel.

```
playwright.inspect('text=Log in')
```

playwright.locator(selector)

Create a locator and query matching elements, for example:

```
playwright.locator('.auth-form', { hasText: 'Log in' });
Locator ()
- element: button
- elements: [button]
```

playwright.selector(element)

Generates selector for the given element. For example, select an element in the Elements panel and pass \$0:

```
playwright.selector($0)  
"div[id='glow-ingress-block'] >> text=/.*Hello.*/"
```

Verbose API logs

Playwright supports verbose logging with the `DEBUG` environment variable.

- Bash
- PowerShell
- Batch

```
DEBUG=pw:api npx playwright test  
$env:DEBUG="pw:api"  
npx playwright test  
set DEBUG=pw:api  
npx playwright test
```

NOTE

For WebKit: launching WebKit Inspector during the execution will prevent the Playwright script from executing any further and will reset pre-configured user agent and device emulation.

Headed mode

Playwright runs browsers in headless mode by default. To change this behavior, use `headless: false` as a launch option.

You can also use the `slowMo` option to slow down execution (by N milliseconds per operation) and follow along while debugging.

```
// Chromium, Firefox, or WebKit  
await chromium.launch({ headless: false, slowMo: 100 });
```

[Previous](#)
[Components \(experimental\)](#)

[Next](#)
[Dialogs](#)

- [VS Code debugger](#)
 - [Error Messages](#)
 - [Live Debugging](#)
 - [Picking a Locator](#)

- [Run in Debug Mode](#)
- [Debug in different Browsers](#)
- [Playwright Inspector](#)
 - [Run in debug mode](#)
 - [Stepping through your tests](#)
 - [Run a test from a specific breakpoint](#)
 - [Live editing locators](#)
 - [Picking locators](#)
 - [Actionability logs](#)
- [Trace Viewer](#)
- [Browser Developer Tools](#)
- [Verbose API logs](#)
- [Headed mode](#)

Learn

- [Getting started](#)
- [Playwright Training](#)
- [Learn Videos](#)
- [Feature Videos](#)

Community

- [Stack Overflow](#)
- [Discord](#)
- [Twitter](#)
- [LinkedIn](#)

More

- [GitHub](#)
- [YouTube](#)
- [Blog](#)
- [Ambassadors](#)

Copyright © 2024 Microsoft