

[Skip to main content](#)



[PlaywrightDocsAPI](#)

[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
- Auto-waiting

On this page

# Auto-waiting

## Introduction

Playwright performs a range of actionability checks on the elements before making actions to ensure these actions behave as expected. It auto-waits for all the relevant checks to pass and only then performs the requested action. If the required checks do not pass within the given timeout, action fails with the `TimeoutError`.

For example, for [locator.click\(\)](#), Playwright will ensure that:

- locator resolves to exactly one element
- element is [Visible](#)
- element is [Stable](#), as in not animating or completed animation
- element [Receives Events](#), as in not obscured by other elements
- element is [Enabled](#)

Here is the complete list of actionability checks performed for each action:

Action	<a href="#">Visible</a>	<a href="#">Stable</a>	<a href="#">Receives Events</a>	<a href="#">Enabled</a>	<a href="#">Editable</a>
<a href="#">locator.check()</a>	Yes	Yes	Yes	Yes	-
<a href="#">locator.click()</a>	Yes	Yes	Yes	Yes	-
<a href="#">locator.dblclick()</a>	Yes	Yes	Yes	Yes	-
<a href="#">locator.setChecked()</a>	Yes	Yes	Yes	Yes	-
<a href="#">locator.tap()</a>	Yes	Yes	Yes	Yes	-
<a href="#">locator.uncheck()</a>	Yes	Yes	Yes	Yes	-
<a href="#">locator.hover()</a>	Yes	Yes	Yes	-	-
<a href="#">locator.dragTo()</a>	Yes	Yes	Yes	-	-
<a href="#">locator.screenshot()</a>	Yes	Yes	-	-	-
<a href="#">locator.fill()</a>	Yes	-	-	Yes	Yes
<a href="#">locator.clear()</a>	Yes	-	-	Yes	Yes
<a href="#">locator.selectOption()</a>	Yes	-	-	Yes	-
<a href="#">locator.selectText()</a>	Yes	-	-	-	-
<a href="#">locator.scrollIntoViewIfNeeded()</a>	-	Yes	-	-	-
<a href="#">locator.blur()</a>	-	-	-	-	-
<a href="#">locator.dispatchEvent()</a>	-	-	-	-	-
<a href="#">locator.focus()</a>	-	-	-	-	-

Action	Visible	Stable	Receives Events	Enabled	Editable
<a href="#">locator.press()</a>	-	-	-	-	-
<a href="#">locator.pressSequentially()</a>	-	-	-	-	-
<a href="#">locator.setInputFiles()</a>	-	-	-	-	-

## Forcing actions

Some actions like [locator.click\(\)](#) support `force` option that disables non-essential actionability checks, for example passing truthy `force` to [locator.click\(\)](#) method will not check that the target element actually receives click events.

## Assertions

Playwright includes auto-retrying assertions that remove flakiness by waiting until the condition is met, similarly to auto-waiting before actions.

Assertion	Description
<a href="#">expect(locator).toBeAttached()</a>	Element is attached
<a href="#">expect(locator).toBeChecked()</a>	Checkbox is checked
<a href="#">expect(locator).toBeDisabled()</a>	Element is disabled
<a href="#">expect(locator).toBeEditable()</a>	Element is editable
<a href="#">expect(locator).toBeEmpty()</a>	Container is empty
<a href="#">expect(locator).toBeEnabled()</a>	Element is enabled
<a href="#">expect(locator).toBeFocused()</a>	Element is focused
<a href="#">expect(locator).toBeHidden()</a>	Element is not visible
<a href="#">expect(locator).toBeInViewport()</a>	Element intersects viewport
<a href="#">expect(locator).toBeVisible()</a>	Element is visible
<a href="#">expect(locator).toContainText()</a>	Element contains text
<a href="#">expect(locator).toHaveAttribute()</a>	Element has a DOM attribute
<a href="#">expect(locator).toHaveClass()</a>	Element has a class property
<a href="#">expect(locator).toHaveCount()</a>	List has exact number of children
<a href="#">expect(locator).toHaveCSS()</a>	Element has CSS property
<a href="#">expect(locator).toHaveId()</a>	Element has an ID
<a href="#">expect(locator).toHaveJSProperty()</a>	Element has a JavaScript property
<a href="#">expect(locator).toHaveText()</a>	Element matches text
<a href="#">expect(locator).toHaveValue()</a>	Input has a value
<a href="#">expect(locator).toHaveValues()</a>	Select has options selected
<a href="#">expect(page).toHaveTitle()</a>	Page has a title
<a href="#">expect(page).toHaveURL()</a>	Page has a URL
<a href="#">expect(response).toBeOK()</a>	Response has an OK status

Learn more in the [assertions guide](#).

## Visible

Element is considered visible when it has non-empty bounding box and does not have `visibility:hidden` computed style.

Note that according to this definition:

- Elements of zero size **are not** considered visible.
- Elements with `display:none` **are not** considered visible.
- Elements with `opacity:0` **are** considered visible.

## Stable

Element is considered stable when it has maintained the same bounding box for at least two consecutive animation frames.

## Enabled

Element is considered enabled unless it is a `<button>`, `<select>`, `<input>` or `<textarea>` with a `disabled` property.

## Editable

Element is considered editable when it is [enabled](#) and does not have `readonly` property set.

## Receives Events

Element is considered receiving pointer events when it is the hit target of the pointer event at the action point. For example, when clicking at the point `(10;10)`, Playwright checks whether some other element (usually an overlay) will instead capture the click at `(10;10)`.

For example, consider a scenario where Playwright will click `Sign Up` button regardless of when the [locator.click\(\)](#) call was made:

- page is checking that user name is unique and `Sign Up` button is disabled;
- after checking with the server, the disabled `Sign Up` button is replaced with another one that is now enabled.

[Previous](#)  
[Authentication](#)

[Next](#)  
[Best Practices](#)

- [Introduction](#)
- [Forcing actions](#)
- [Assertions](#)

- [Visible](#)
- [Stable](#)
- [Enabled](#)
- [Editable](#)
- [Receives Events](#)

## Learn

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

## Community

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

## More

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

Copyright © 2024 Microsoft