puppeteer TS

16.1.1 • Public • Published 2 days ago

Readme

Explore BETA

12 Dependencies

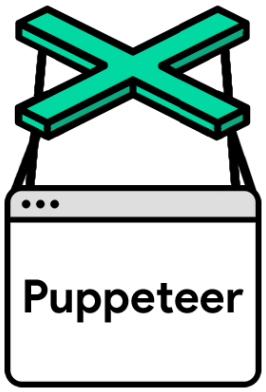
5,203 Dependents

758 Versions

Puppeteer

CI failing npm v16.1.1

API | FAQ | Contributing | Troubleshooting



> Puppeteer is a Node library which provides a high-level API to control Chrome or Chromium over the **DevTools Protocol**. Puppeteer runs **headless** by default, but can be configured to run full (non-headless) Chrome or Chromium.

What can I do?

Most things that you can do manually in the browser can be done using Puppeteer! Here are a few examples to get you started:

- Generate screenshots and PDFs of pages.
- Crawl a SPA (Single-Page Application) and generate pre-rendered content (i.e. "SSR" (Server-Side Rendering)).
- Automate form submission, UI testing, keyboard input, etc.
- Create an up-to-date, automated testing environment. Run your tests directly in the latest version of Chrome using the latest JavaScript and browser features.

- Capture a timeline trace of your site to help diagnose performance issues.
- Test Chrome Extensions.

Getting Started

Installation

To use Puppeteer in your project, run:

```
npm i puppeteer
# or "yarn add puppeteer"
```

When you install Puppeteer, it downloads a recent version of Chromium (~170MB Mac, ~282MB Linux, ~280MB Win) that is guaranteed to work with the API (customizable through **Environment Variables**). For a version of Puppeteer purely for connection, see `puppeteer-core`.

Environment Variables

Puppeteer looks for certain **environment variables** to aid its operations. If Puppeteer doesn't find them in the environment during the installation step, a lowercased variant of these variables will be used from the **npm config**.

- `HTTP_PROXY`, `HTTPS_PROXY`, `NO_PROXY` - defines HTTP proxy settings that are used to download and run the browser.
- `PUPPETEER_SKIP_CHROMIUM_DOWNLOAD` - do not download bundled Chromium during installation step.
- `PUPPETEER_TMP_DIR` - defines the directory to be used by Puppeteer for creating temporary files. Defaults to `os.tmpdir()`.
- `PUPPETEER_DOWNLOAD_HOST` - overwrite URL prefix that is used to download Chromium. Note: this includes protocol and might even include path prefix. Defaults to `https://storage.googleapis.com`.
- `PUPPETEER_DOWNLOAD_PATH` - overwrite the path for the downloads folder. Defaults to `<root>/.local-chromium`, where `<root>` is Puppeteer's package root.
- `PUPPETEER_CHROMIUM_REVISION` - specify a certain version of Chromium you'd like Puppeteer to use. See `puppeteer.launch` on how executable path is inferred.
- `PUPPETEER_EXECUTABLE_PATH` - specify an executable path to be used in `puppeteer.launch`.
- `PUPPETEER_PRODUCT` - specify which browser you'd like Puppeteer to use. Must be one of `chrome` or `firefox`. This can also be used during installation to fetch the recommended browser binary. Setting `product` programmatically in `puppeteer.launch` supersedes this environment variable. The product is exposed in `puppeteer.product`
- `PUPPETEER_EXPERIMENTAL_CHROMIUM_MAC_ARM` — specify Puppeteer download Chromium for Apple M1. On Apple M1 devices Puppeteer by default downloads the version for

Intel's processor which runs via Rosetta. It works without any problems, however, with this option, you should get more efficient resource usage (CPU and RAM) that could lead to a faster execution time.

:::danger

Puppeteer is only **guaranteed to work** with the bundled Chromium, use at your own risk.

:::

:::caution

`PUPPETEER_*` env variables are not accounted for in `puppeteer-core`.

:::

puppeteer-core

Every release since v1.7.0 we publish two packages:

- `puppeteer`
- `puppeteer-core`

`puppeteer` is a *product* for browser automation. When installed, it downloads a version of Chromium, which it then drives using `puppeteer-core`. Being an end-user product, `puppeteer` supports a bunch of convenient `PUPPETEER_*` env variables to tweak its behavior.

`puppeteer-core` is a *library* to help drive anything that supports DevTools protocol.
`puppeteer-core` doesn't download Chromium when installed. Being a library, `puppeteer-core` is fully driven through its programmatic interface and disregards all the `PUPPETEER_*` env variables.

To sum up, the only differences between `puppeteer-core` and `puppeteer` are:

- `puppeteer-core` doesn't automatically download Chromium when installed.
- `puppeteer-core` ignores all `PUPPETEER_*` env variables.

In most cases, you'll be fine using the `puppeteer` package.

However, you should use `puppeteer-core` if:

- you're building another end-user product or library atop of DevTools protocol. For example, one might build a PDF generator using `puppeteer-core` and write a custom `install.js` script that downloads `headless_shell` instead of Chromium to save disk space.
- you're bundling Puppeteer to use in Chrome Extension / browser with the DevTools protocol where downloading an additional Chromium binary is unnecessary.

- you're building a set of tools where `puppeteer-core` is one of the ingredients and you want to postpone `install.js` script execution until Chromium is about to be used.

When using `puppeteer-core`, remember to change the *include* line:

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

You will then need to call **`puppeteer.connect`** or **`puppeteer.launch`** with an explicit `executablePath` or `channel` option.

Usage

Puppeteer follows the latest **maintenance LTS** version of Node.

Puppeteer will be familiar to people using other browser testing frameworks. You create an instance of `Browser`, open pages, and then manipulate them with **Puppeteer's API**.

Example - navigating to **https://example.com** and saving a screenshot as *example.png*:

Save file as **example.js**

```
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: 'example.png'});

  await browser.close();
})();
```

Execute script on the command line

```
node example.js
```

Puppeteer sets an initial page size to 800×600px, which defines the screenshot size. The page size can be customized with **`Page.setViewport()`**.

Example - create a PDF.

Save file as **hn.js**

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

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://news.ycombinator.com', {
    waitUntil: 'networkidle2',
  });
  await page.pdf({path: 'hn.pdf', format: 'a4'});

  await browser.close();
})();
```

Execute script on the command line

```
node hn.js
```

See **`Page.pdf`** for more information about creating pdfs.

Example - evaluate script in the context of the page

Save file as **get-dimensions.js**

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

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');

  // Get the "viewport" of the page, as reported by the page.
  const dimensions = await page.evaluate(() => {
    return {
      width: document.documentElement.clientWidth,
      height: document.documentElement.clientHeight,
      deviceScaleFactor: window.devicePixelRatio,
    };
  });
```

```
  console.log('Dimensions:', dimensions);

  await browser.close();
})();
```

Execute script on the command line

```
node get-dimensions.js
```

See `Page.evaluate` and related methods like `Page.evaluateOnNewDocument` and `Page.exposeFunction`.

Running in Docker

Puppeteer offers a Docker image that includes Chromium along with the required dependencies and a pre-installed Puppeteer version. The image is available via the **GitHub Container Registry**. The latest image is tagged as `latest` and other tags match Puppeteer versions. For example,

```
docker pull ghcr.io/puppeteer/puppeteer:latest # pulls the latest
docker pull ghcr.io/puppeteer/puppeteer:16.1.0 # pulls the image that
```

The image is meant for running the browser in the sandbox mode and therefore, running the image requires the `SYS_ADMIN` capability. For example,

```
docker run -i --init --cap-add=SYS_ADMIN --rm ghcr.io/puppeteer/puppe
```

Replace the path to `smoke-test.js` with a path to your script. The script can import or require the `puppeteer` module because it's pre-installed inside the image.

Currently, the image includes the LTS version of Node.js. If you need to build an image based on a different base image, you can use our `Dockerfile` as the starting point.

Working with Chrome Extensions

Puppeteer can be used for testing Chrome Extensions.

:::caution

Extensions in Chrome / Chromium currently only work in non-headless mode and experimental Chrome headless mode.

:::

The following is code for getting a handle to the **background page** of an extension whose source is located in `./my-extension` :

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

(async () => {
  const pathToExtension = require('path').join(__dirname, 'my-extensi
  const browser = await puppeteer.launch({
    headless: 'chrome',
    args: [
      `--disable-extensions-except=${pathToExtension}`,
      `--load-extension=${pathToExtension}`,
    ],
  });
  const backgroundPageTarget = await browser.waitForTarget(
    target => target.type() === 'background_page'
  );
  const backgroundPage = await backgroundPageTarget.page();
  // Test the background page as you would any other page.
  await browser.close();
})();
```

:::note

Chrome Manifest V3 extensions have a background ServiceWorker of type 'service_worker', instead of a page of type 'background_page'.

:::

:::note

It is not yet possible to test extension popups or content scripts.

:::

Default runtime settings

1. Uses Headless mode

Puppeteer launches Chromium in **headless mode**. To launch a full version of Chromium, set the **`headless option`** when launching a browser:

```
const browser = await puppeteer.launch({headless: false}); // default
```

2. Runs a bundled version of Chromium

By default, Puppeteer downloads and uses a specific version of Chromium so its API is guaranteed to work out of the box. To use Puppeteer with a different version of Chrome or Chromium, pass in the executable's path when creating a `Browser` instance:

```
const browser = await puppeteer.launch({executablePath: '/path/to/Chr
```

You can also use Puppeteer with Firefox Nightly (experimental support). See **`Puppeteer.launch`** for more information.

See **`this article`** for a description of the differences between Chromium and Chrome. **`This article`** describes some differences for Linux users.

3. Creates a fresh user profile

Puppeteer creates its own browser user profile which it **cleans up on every run**.

Resources

- API Documentation
- Examples
- Community list of Puppeteer resources

Debugging tips

1. Turn off headless mode - sometimes it's useful to see what the browser is displaying. Instead of launching in headless mode, launch a full version of the browser using `headless: false`:

   ```
   const browser = await puppeteer.launch({headless: false});
   ```

2. Slow it down - the `slowMo` option slows down Puppeteer operations by the specified amount of milliseconds. It's another way to help see what's going on.

```
const browser = await puppeteer.launch({
  headless: false,
  slowMo: 250, // slow down by 250ms
});
```

3. Capture console output - You can listen for the `console` event. This is also handy when debugging code in `page.evaluate()`:

```
page.on('console', msg => console.log('PAGE LOG:', msg.text()));

await page.evaluate(() => console.log(`url is ${location.href}`));
```

4. Use debugger in application code browser

   There are two execution context: node.js that is running test code, and the browser running application code being tested. This lets you debug code in the application code browser; ie code inside `evaluate()`.

   - Use `{devtools: true}` when launching Puppeteer:

     ```
     const browser = await puppeteer.launch({devtools: true});
     ```

   - Change default test timeout:

     jest: `jest.setTimeout(100000);`

     jasmine: `jasmine.DEFAULT_TIMEOUT_INTERVAL = 100000;`

     mocha: `this.timeout(100000);` (don't forget to change test to use function and not '=>')

   - Add an evaluate statement with `debugger` inside / add `debugger` to an existing evaluate statement:

     ```
     await page.evaluate(() => {
       debugger;
     });
     ```

     The test will now stop executing in the above evaluate statement, and chromium will stop in debug mode.

5. Use debugger in node.js

   This will let you debug test code. For example, you can step over `await page.click()` in the node.js script and see the click happen in the application code browser.

   Note that you won't be able to run `await page.click()` in DevTools console due to this **Chromium bug**. So if you want to try something out, you have to add it to your test file.

   - Add `debugger;` to your test, eg:

     ```
     debugger;
     await page.click('a[target=_blank]');
     ```

   - Set `headless` to `false`
   - Run `node --inspect-brk` , eg `node --inspect-brk node_modules/.bin/jest tests`
   - In Chrome open `chrome://inspect/#devices` and click `inspect`
   - In the newly opened test browser, type `F8` to resume test execution
   - Now your `debugger` will be hit and you can debug in the test browser

6. Enable verbose logging - internal DevTools protocol traffic will be logged via the **`debug`** module under the `puppeteer` namespace.

   ```
   # Basic verbose logging
   env DEBUG="puppeteer:*" node script.js

   # Protocol traffic can be rather noisy. This example filters out all N
   env DEBUG="puppeteer:*" env DEBUG_COLORS=true node script.js 2>&1 | gr
   ```

7. Debug your Puppeteer (node) code easily, using **ndb**

- `npm install -g ndb` (or even better, use npx!)
- add a `debugger` to your Puppeteer (node) code
- add `ndb` (or `npx ndb`) before your test command. For example:

  `ndb jest` or `ndb mocha` (or `npx ndb jest` / `npx ndb mocha`)
- debug your test inside chromium like a boss!

Contributing

Check out our **contributing guide** to get an overview of Puppeteer development.

FAQ

Our **FAQ** has migrated to **our site**.

Keywords

puppeteer **chrome** **headless** **automation**

Install

```
$ npm i puppeteer
```

Repository

github.com/puppeteer/puppeteer

Homepage

github.com/puppeteer/puppeteer#readme

Weekly Downloads

3,565,226

Version	License
16.1.1	Apache-2.0

Unpacked Size	Total Files
3.4 MB	555

Last publish

2 days ago

Collaborators

Collaborators

Try on RunKit

Report malware

Support

Help

Advisories

Status

Contact npm

Company

About

Blog

Press

Terms & Policies

Policies

Terms of Use

Code of Conduct

Privacy