westy92 / **html-pdf-chrome**

HTML to PDF converter via Chrome/Chromium

#chrome #chromium #html #html-pdf-chrome #pdf #pdf-generation #typescript #javascript #headless-chrome #headless-chromium #headless-browsers #headless #windows #macos #linux #nodejs #node-js #google #google-chrome #pdf-generator

111 commits | **4** branches | **9** releases | **3** contributors | MIT

Branch: master | New pull request | Find file | Clone or download

westy92 Split Chrome typings into separate files. Latest commit `314d6c7` 4 days ago

.vscode	Add CI (#1)	6 months ago
src	Split Chrome typings into separate files.	4 days ago
test	Split Chrome typings into separate files.	4 days ago
.appveyor.yml	Test on Node.js 9.	5 days ago
.gitignore	Generate js on publish.	4 months ago
.istanbul.yml	Fix istanbul.	6 months ago
.npmignore	Add .istanbul.yml to .npmignore.	6 months ago
.travis.yml	Test on Node.js 9.	5 days ago
LICENSE	Initial commit	6 months ago
README.md	Add option for clearing Chrome's cache.	5 days ago
gulpfile.js	Add unit test retries.	5 days ago
package.json	Release v0.4.1.	4 days ago
tsconfig.json	Initial commit.	6 months ago
tslint.json	Check for unused variables.	4 months ago

README.md

html-pdf-chrome

npm package 0.4.1 | build passing | build passing | codecov 100% | dependencies up to date | vulnerabilities 0

HTML to PDF converter via Chrome/Chromium.

Prerequisites

- Latest Chrome/Chromium (latest recommended, 59 or higher required but some features may not work)

- Windows, macOS, or Linux
- Node.js v6 or later

Installation

```
npm install --save html-pdf-chrome
```

Usage

Note: It is *strongly* recommended that you keep Chrome running side-by-side with Node.js. There is significant overhead starting up Chrome for each PDF generation which can be easily avoided.

It's suggested to use pm2 to ensure Chrome continues to run. If it crashes, it will restart automatically.

As of this writing, headless Chrome uses about 65mb of RAM while idle.

```
# install pm2 globally
npm install -g pm2
# start Chrome and be sure to specify a port to use in the html-pdf-chrome options.
pm2 start google-chrome \
  --interpreter none \
  -- \
  --headless \
  --disable-gpu \
  --disable-translate \
  --disable-extensions \
  --disable-background-networking \
  --safebrowsing-disable-auto-update \
  --disable-sync \
  --metrics-recording-only \
  --disable-default-apps \
  --no-first-run \
  --mute-audio \
  --hide-scrollbars \
  --remote-debugging-port=<port goes here>
# run your Node.js app.
```

TypeScript:

```
import * as htmlPdf from 'html-pdf-chrome';

const html = '<p>Hello, world!</p>';
const options: htmlPdf.CreateOptions = {
  port: 9222, // port Chrome is listening on
};

// async
const pdf = await htmlPdf.create(html, options);
await pdf.toFile('test.pdf');
const base64 = pdf.toBase64();
const buffer = pdf.toBuffer();

// Promise
htmlPdf.create(html, options).then((pdf) => pdf.toFile('test.pdf'));
htmlPdf.create(html, options).then((pdf) => pdf.toBase64());
htmlPdf.create(html, options).then((pdf) => pdf.toBuffer());
```

JavaScript:

```
const htmlPdf = require('html-pdf-chrome');
const html = '<p>Hello, world!</p>';
const options = {
  port: 9222, // port Chrome is listening on
};

htmlPdf.create(html, options).then((pdf) => pdf.toFile('test.pdf'));
htmlPdf.create(html, options).then((pdf) => pdf.toBase64());
htmlPdf.create(html, options).then((pdf) => pdf.toBuffer());
```

View the full documentation in the source code.

Using an External Site

```
import * as htmlPdf from 'html-pdf-chrome';

const options: htmlPdf.CreateOptions = {
  port: 9222, // port Chrome is listening on
};

const url = 'https://github.com/westy92/html-pdf-chrome';
const pdf = await htmlPdf.create(url, options);
```

Using a Template Engine

Pug (formerly known as Jade)

```
import * as htmlPdf from 'html-pdf-chrome';
import * as pug from 'pug';

const template = pug.compile('p Hello, #{noun}!');
const templateData = {
  noun: 'world',
};
const options: htmlPdf.CreateOptions = {
  port: 9222, // port Chrome is listening on
};

const html = template(templateData);
const pdf = await htmlPdf.create(html, options);
```

Trigger Render Completion

There are a few `CompletionTrigger` types that wait for something to occur before triggering PDF printing.

- Callback - waits for a callback to be called
- Element - waits for an element to be injected into the DOM
- Event - waits for an Event to fire
- Timer - waits a specified amount of time
- Variable - waits for a variable to be set to `true`
- Custom - extend `htmlPdf.CompletionTrigger.CompletionTrigger`

```
const options: htmlPdf.CreateOptions = {
  port: 9222, // port Chrome is listening on
  completionTrigger: new htmlPdf.CompletionTrigger.Timer(5000), // milliseconds
};

// Alternative completionTrigger options:
new htmlPdf.CompletionTrigger.Callback(
  'cbName', // optional, name of the callback to define for the browser to call when finished rendering.  I
  5000 // optional, timeout (milliseconds)
),
new htmlPdf.CompletionTrigger.Element(
  'div#myElement', // name of the DOM element to wait for
  5000 // optional, timeout (milliseconds)
),

new htmlPdf.CompletionTrigger.Event(
  'myEvent', // name of the event to listen for
  '#myElement', // optional DOM element CSS selector to listen on, defaults to body
  5000 // optional timeout (milliseconds)
),

new htmlPdf.CompletionTrigger.Variable(
  'myVarName', // optional, name of the variable to wait for.  Defaults to 'htmlPdfDone'
  5000 // optional, timeout (milliseconds)
),
```

License

html-pdf-chrome is released under the MIT License.