eKoopmans / **html2pdf.js**

Client-side HTML-to-PDF rendering using pure JS.

#javascript #pdf-generation #html #client-side #canvas

210 commits | **8** branches | **0** packages | **21** releases | **4** contributors | MIT

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

eKoopmans Merge pull request #238 from oschwede/fix/sandbox-promises ··· Latest commit `db15497` on 8 Jul

dist	Build v0.9.1	last year
src	Merge pull request #238 from oschwede/fix/sandbox-promises	5 months ago
test/manual	Add advanced tests of avoid-all	last year
.babelrc	Add object.assign transpiling for IE	2 years ago
.gitignore	Update gitignore	2 years ago
.npmignore	Add npmignore to reduce npm package size	2 years ago
LICENSE	Move the license	2 years ago
README.md	Remove develop branch	6 months ago
_config.yml	Set theme jekyll-theme-cayman	10 months ago
bower.json	Add Bower config	2 years ago
gulpfile.js	Update gulpfile for 4.0 compatibility	last year
package-lock.json	Add minimist for gulp argument parsing	last year
package.json	Update repo and homepage urls	6 months ago
rollup.config.js	Add es6-promise to globals in rollup config.	5 months ago

README.md

html2pdf.js

html2pdf.js converts any webpage or element into a printable PDF entirely client-side using html2canvas and jsPDF.

Table of contents

- Getting started
  - HTML
  - NPM
  - Bower
  - Console
- Usage
  - Advanced usage
    - Workflow
    - Worker API
- Options
  - Page-breaks
    - Page-break settings
    - Page-break modes
    - Example usage
  - Image type and quality
- Progress tracking
- Dependencies
- Contributing
  - Issues
  - Tests
  - Pull requests
- Credits
- License

Getting started

HTML

The simplest way to use html2pdf.js is to download `dist/html2pdf.bundle.min.js` to your project folder and include it in your HTML with:

```
<script src="html2pdf.bundle.min.js"></script>
```

Note: Click here for more information about using the unbundled version `dist/html2canvas.min.js`.

NPM

Install html2pdf.js and its dependencies using NPM with `npm install --save html2pdf.js` (make sure to include `.js` in the package name).

*Note: You can use NPM to create your project, but html2pdf.js **will not run in Node.js**, it must be run in a browser.*

Bower

Install html2pdf.js and its dependencies using Bower with `bower install --save html2pdf.js` (make sure to include `.js` in the package name).

Console

If you're on a webpage that you can't modify directly and wish to use html2pdf.js to capture a screenshot, you can follow these steps:

1. Open your browser's console (instructions for different browsers here).

2. Paste in this code:

```
function addScript(url) {
    var script = document.createElement('script');
    script.type = 'application/javascript';
    script.src = url;
    document.head.appendChild(script);
}
addScript('https://raw.githack.com/eKoopmans/html2pdf/master/dist/html2pdf.bundle.js');
```

3. You may now execute html2pdf.js commands directly from the console. To capture a default PDF of the entire page, use `html2pdf(document.body)` .

Usage

Once installed, html2pdf.js is ready to use. The following command will generate a PDF of `#element-to-print` and prompt the user to save the result:

```
var element = document.getElementById('element-to-print');
html2pdf(element);
```

Advanced usage

Every step of html2pdf.js is configurable, using its new Promise-based API. If html2pdf.js is called without arguments, it will return a `Worker` object:

```
var worker = html2pdf();  // Or:  var worker = new html2pdf.Worker;
```

This worker has methods that can be chained sequentially, as each Promise resolves, and allows insertion of your own intermediate functions between steps. A prerequisite system allows you to skip over mandatory steps (like canvas creation) without any trouble:

```
// This will implicitly create the canvas and PDF objects before saving.
var worker = html2pdf().from(element).save();
```

Workflow

The basic workflow of html2pdf.js tasks (enforced by the prereq system) is:

```
.from() -> .toContainer() -> .toCanvas() -> .toImg() -> .toPdf() -> .save()
```

Worker API

Method	Arguments	Description
from	src, type	Sets the source (HTML string or element) for the PDF. Optional `type` specifies other sources: `'string'` , `'element'` , `'canvas'` , or `'img'` .
to	target	Converts the source to the specified target (`'container'` , `'canvas'` , `'img'` , or `'pdf'`). Each target also has its own `toX` method that can be called directly: `toContainer()` , `toCanvas()` , `toImg()` , and `toPdf()` .
output	type, options, src	Routes to the appropriate `outputPdf` or `outputImg` method based on specified `src` (`'pdf'` (default) or `'img'`).

Method	Arguments	Description
outputPdf	type, options	Sends `type` and `options` to the jsPDF object's `output` method, and returns the result as a Promise (use `.then` to access). See the jsPDF source code for more info.
outputImg	type, options	Returns the specified data type for the image as a Promise (use `.then` to access). Supported types: `'img'`, `'datauristring'` / `'dataurlstring'`, and `'datauri'` / `'dataurl'`.
save	filename	Saves the PDF object with the optional filename (creates user download prompt).
set	opt	Sets the specified properties. See Options below for more details.
get	key, cbk	Returns the property specified in `key`, either as a Promise (use `.then` to access), or by calling `cbk` if provided.
then	onFulfilled, onRejected	Standard Promise method, with `this` re-bound to the Worker, and with added progress-tracking (see Progress below). Note that `.then` returns a `Worker`, which is a subclass of Promise.
thenCore	onFulFilled, onRejected	Standard Promise method, with `this` re-bound to the Worker (no progress-tracking). Note that `.thenCore` returns a `Worker`, which is a subclass of Promise.
thenExternal	onFulfilled, onRejected	True Promise method. Using this 'exits' the Worker chain - you will not be able to continue chaining Worker methods after `.thenExternal`.
catch, catchExternal	onRejected	Standard Promise method. `catchExternal` exits the Worker chain - you will not be able to continue chaining Worker methods after `.catchExternal`.
error	msg	Throws an error in the Worker's Promise chain.

A few aliases are also provided for convenience:

Method	Alias
save	saveAs
set	using
output	export
then	run

Options

html2pdf.js can be configured using an optional `opt` parameter:

```
var element = document.getElementById('element-to-print');
var opt = {
  margin:       1,
  filename:     'myfile.pdf',
  image:        { type: 'jpeg', quality: 0.98 },
  html2canvas:  { scale: 2 },
  jsPDF:        { unit: 'in', format: 'letter', orientation: 'portrait' }
};

// New Promise-based usage:
html2pdf().set(opt).from(element).save();

// Old monolithic-style usage:
html2pdf(element, opt);
```

The `opt` parameter has the following optional fields:

Name	Type	Default	Description
margin	number or array	`0`	PDF margin (in jsPDF units). Can be a single number, `[vMargin, hMargin]`, or `[top, left, bottom, right]`.
filename	string	`'file.pdf'`	The default filename of the exported PDF.
pagebreak	object	`{mode: ['css', 'legacy']}`	Controls the pagebreak behaviour on the page. See Page-breaks below.
image	object	`{type: 'jpeg', quality: 0.95}`	The image type and quality used to generate the PDF. See Image type and quality below.
enableLinks	boolean	`true`	If enabled, PDF hyperlinks are automatically added ontop of all anchor tags.
html2canvas	object	`{ }`	Configuration options sent directly to `html2canvas` (see here for usage).
jsPDF	object	`{ }`	Configuration options sent directly to `jsPDF` (see here for usage).

Page-breaks

html2pdf.js has the ability to automatically add page-breaks to clean up your document. Page-breaks can be added by CSS styles, set on individual elements using selectors, or avoided from breaking inside all elements (`avoid-all` mode).

By default, html2pdf.js will respect most CSS `break-before`, `break-after`, and `break-inside` rules, and also add page-breaks after any element with class `html2pdf__page-break` (for legacy purposes).

Page-break settings

Setting	Type	Default	Description
mode	string or array	`['css', 'legacy']`	The mode(s) on which to automatically add page-breaks. One or more of `'avoid-all'`, `'css'`, and `'legacy'`.
before	string or array	`[]`	CSS selectors for which to add page-breaks before each element. Can be a specific element with an ID (`'#myID'`), all elements of a type (e.g. `'img'`), all of a class (`'.myClass'`), or even `'*'` to match every element.
after	string or array	`[]`	Like 'before', but adds a page-break immediately after the element.
avoid	string or array	`[]`	Like 'before', but avoids page-breaks on these elements. You can enable this feature on every element using the 'avoid-all' mode.

Page-break modes

Mode	Description
avoid-all	Automatically adds page-breaks to avoid splitting any elements across pages.
css	Adds page-breaks according to the CSS `break-before`, `break-after`, and `break-inside` properties. Only recognizes `always/left/right` for before/after, and `avoid` for inside.

Mode	Description
legacy	Adds page-breaks after elements with class `html2pdf__page-break`. This feature may be removed in the future.

Example usage

```
// Avoid page-breaks on all elements, and add one before #page2el.
html2pdf().set({
  pagebreak: { mode: 'avoid-all', before: '#page2el' }
});

// Enable all 'modes', with no explicit elements.
html2pdf().set({
  pagebreak: { mode: ['avoid-all', 'css', 'legacy'] }
});

// No modes, only explicit elements.
html2pdf().set({
  pagebreak: { before: '.beforeClass', after: ['#after1', '#after2'], avoid: 'img' }
});
```

Image type and quality

You may customize the image type and quality exported from the canvas by setting the `image` option. This must be an object with the following fields:

Name	Type	Default	Description
type	string	'jpeg'	The image type. HTMLCanvasElement only supports 'png', 'jpeg', and 'webp' (on Chrome).
quality	number	0.95	The image quality, from 0 to 1. This setting is only used for jpeg/webp (not png).

These options are limited to the available settings for HTMLCanvasElement.toDataURL(), which ignores quality settings for 'png' images. To enable png image compression, try using the canvas-png-compression shim, which should be an in-place solution to enable png compression via the `quality` option.

Progress tracking

The Worker object returned by `html2pdf()` has a built-in progress-tracking mechanism. It will be updated to allow a progress callback that will be called with each update, however it is currently a work-in-progress.

Dependencies

html2pdf.js depends on the external packages html2canvas, jsPDF, and es6-promise. These dependencies are automatically loaded when using NPM or the bundled package.

If using the unbundled `dist/html2pdf.min.js` (or its un-minified version), you must also include each dependency. Order is important, otherwise html2canvas will be overridden by jsPDF's own internal implementation:

```
<script src="es6-promise.auto.min.js"></script>
<script src="jspdf.min.js"></script>
<script src="html2canvas.min.js"></script>
<script src="html2pdf.min.js"></script>
```

Contributing

Issues

When submitting an issue, please provide reproducible code that highlights the issue, preferably by creating a fork of this template jsFiddle (which has html2pdf.js already loaded). Remember that html2pdf.js uses html2canvas and jsPDF as dependencies, so it's a good idea to check each of those repositories' issue trackers to see if your problem has already been addressed.

Known issues

1. **Rendering:** The rendering engine html2canvas isn't perfect (though it's pretty good!). If html2canvas isn't rendering your content correctly, I can't fix it.

   - You can test this with something like this fiddle, to see if there's a problem in the canvas creation itself.

2. **Node cloning (CSS etc):** The way html2pdf.js clones your content before sending to html2canvas is buggy. A fix is currently being developed - try out:

   - direct file: Go to html2pdf.js/bugfix/clone-nodes-BUILD and replace the files in your project with the relevant files (e.g. `dist/html2pdf.bundle.js`)
   - npm: `npm install eKoopmans/html2pdf.js#bugfix/clone-nodes-BUILD`
   - Related project: Bugfix: Cloned nodes

3. **Resizing:** Currently, html2pdf.js resizes the root element to fit onto a PDF page (causing internal content to "reflow").

   - This is often desired behaviour, but not always.
   - There are plans to add alternate behaviour (e.g. "shrink-to-page"), but nothing that's ready to test yet.
   - Related project: Feature: Single-page PDFs

4. **Rendered as image:** html2pdf.js renders all content into an image, then places that image into a PDF.

   - This means text is *not selectable or searchable*, and causes large file sizes.
   - This is currently unavoidable, however recent improvements in jsPDF mean that it may soon be possible to render straight into vector graphics.
   - Related project: Feature: New renderer

5. **Promise clashes:** html2pdf.js relies on specific Promise behaviour, and can fail when used with custom Promise libraries.

   - In the next release, Promises will be sandboxed in html2pdf.js to remove this issue.
   - Related project: Bugfix: Sandboxed promises

6. **Maximum size:** HTML5 canvases have a maximum height/width. Anything larger will fail to render.

   - This is a limitation of HTML5 itself, and results in large PDFs rendering completely blank in html2pdf.js.
   - The jsPDF canvas renderer (mentioned in Known Issue #4) may be able to fix this issue!
   - Related project: Bugfix: Maximum canvas size

Tests

html2pdf.js is currently sorely lacking in unit tests. Any contributions or suggestions of automated (or manual) tests are welcome. This is high on the to-do list for this project.

Pull requests

If you want to create a new feature or bugfix, please feel free to fork and submit a pull request! Create a fork, branch off of `master`, and make changes to the `/src/` files (rather than directly to `/dist/`). You can test your changes by rebuilding with `npm run build`.

Credits

Erik Koopmans

Contributors

- @WilcoBreedt
- @Ranger1230

Special thanks

- Sauce Labs for unit testing.

License

The MIT License