Node.js

The npm package `weblitzpdf` is Weblitz for Node.js: HTML, a file or a web address to a PDF, an Excel, Word or PowerPoint file, or a picture, printed as Chrome prints them, with no browser to install, start or drive. The same package renders RDLC and RDL reports and Markdown. Its options are Puppeteer's `page.pdf()` options under the same names.

Install

```
npm install weblitzpdf
```

Version 2.0.0-rc.2, a prerelease of 2.0. Node.js 22 or later, on Linux with glibc (x64, Arm64), macOS (Arm64, x64) and Windows (x64, Arm64). npm installs the engine for your platform only, as an optional dependency (one of @weblitzpdf/linux-x64, @weblitzpdf/linux-arm64, @weblitzpdf/darwin-arm64, @weblitzpdf/darwin-x64, @weblitzpdf/win32-x64, @weblitzpdf/win32-arm64): it holds the engine and the .NET runtime it runs on, so there is nothing else to install and no .NET on the machine is needed.

- Installing with `--omit=optional` leaves the engine out, and so does a `node_modules` copied from another platform (a Mac's into a Linux container): run `npm ci` in the image. A call then fails with the code `workerMissing`, naming the package it misses.
- Alpine and other musl systems are not supported yet: use a glibc image, `node:22-bookworm-slim` for example.
- On Linux the engine needs fontconfig and ICU, which slim images lack: `apt-get install -y --no-install-recommends libfontconfig1 libicu72` (the ICU package's number follows the system's release).

Your first PDFs

```
import { readFile, writeFile } from 'node:fs/promises';
import { pathToFileURL } from 'node:url';
import { renderHtml, renderUrl } from 'weblitzpdf';   // or: const { renderHtml,
renderUrl } = require('weblitzpdf');

// HTML text
const invoice = await renderHtml('<h1>Invoice 42</h1><p>...</p>', { format: 'A4',
printBackground: true });
await writeFile('invoice.pdf', invoice.file);

// A web page, after its scripts ran; path writes the file too, as Puppeteer's
await renderUrl('https://example.com/report', {
  path: 'report.pdf',
  margin: { top: '18mm', right: '18mm', bottom: '20mm', left: '20mm' },
});

// A file on disk: its CSS, pictures, fonts and scripts are read from its folder
const letters = await renderHtml(await readFile('templates/letters.html', 'utf8'), {
  baseUrl: pathToFileURL('templates/').href,
});

// What could not be drawn or loaded is said, never thrown
for (const note of invoice.notes) console.log(`${note.severity} ${note.code}:
${note.message}`);
```

A render answers `{ file, contentType, extension, pages, notes, elapsedMs, trial }`, `file` a Buffer. Beside Puppeteer's options (`format`, `width`, `height`, `landscape`, `margin`, `preferCSSPageSize`, `printBackground`, `scale`, `pageRanges`, `displayHeaderFooter`, `headerTemplate`, `footerTemplate`, `path`) the engine has its own: `baseUrl`, `output`, `scripts`, `network`, `title`, `author`, `pdfA`, `compression`, `locale`, `timeZone`, `now`, `useInstalledFonts`, `styles`, `hide`, `consent: 'hide'` and `ads: 'block'`. They are .NET's options under JavaScript's names.

Excel, Word, PowerPoint and pictures

```
const workbook = await renderUrl('https://example.com/prices', { output: 'xlsx' });
// or 'docx', 'pptx'
const shot = await renderUrl('https://example.com', { output: 'png', viewport: { width:
1280, height: 720, deviceScaleFactor: 2 } });
const pages = await renderHtml(html, { output: 'png', capture: 'pages', format: 'A4'
});   // pages.file is a ZIP
```

`output: 'png'` (or `'jpeg'`, `'webp'`, `'tiff'`) draws the page instead of printing it: pictures of a page. The Office files are described in Excel, Word and PowerPoint.

What a page loads

A page loads nothing from the network by default but, for `renderUrl`, its own host. `network` opens more, with headers, cookies and credentials for pages behind a login; each goes only to the hosts it names (`allowedHosts: ['*']` allows any host):

```
await renderUrl('https://app.example.com/reports/42', {
  network: {
    allowedHosts: ['app.example.com', 'cdn.example.com'],
    headers: [{ name: 'Authorization', value: `Bearer ${token}`, hosts:
['app.example.com'] }],
    cookies: [{ name: 'session', value: sessionId, domain: 'app.example.com', secure:
true, httpOnly: true }],
  },
});
```

Reports and Markdown

The same package renders SSRS2's reports: `renderReport`, `checkReport`, `markdownTo` and `importReport`. `@ssrs2/ssrs2` (`npm install @ssrs2/ssrs2`) has the same functions for projects that only render reports; it runs on `weblitzpdf` at the same version.

```
import { renderReport, checkReport } from 'weblitzpdf';

const pdf = await renderReport({ path: 'Reports/Invoice.rdlc' }, {
  format: 'PDF',   // or WORDOPENXML, EXCELOPENXML, PPTX, IMAGE, PNG, HTML5, CSV, XML
(docx, xlsx...)
  datasets: { Items: [{ Description: 'Widget', Price: 104.99, Qty: 1 }] },
  parameters: { InvoiceNumber: '42' },
});

const check = await checkReport({ path: 'Reports/Invoice.rdlc' });   // its problems,
without rendering it
if (!check.ok) console.log(check.notes);
```

Workers, timeouts and errors

The engine runs in worker processes, started on the first call and kept warm; idle workers do not keep Node.js running. `configure()` sets the shared pool's options before that, and `new Engine(options)` makes another pool, with the same methods and `close()`.

```
import { configure, Engine } from 'weblitzpdf';

configure({ workers: 2, timeout: 30_000 });   // milliseconds

const engine = new Engine({ workers: 4, concurrency: 2, maxQueue: 100 });
const result = await engine.renderHtml(html, { timeout: 5_000, signal:
AbortSignal.timeout(10_000) });
await engine.close();
```

A failed call throws a `WeblitzError` whose `code` says why: `badRequest`, `tooLarge`, `timeout`, `refused`, `invalidReport`, `renderFailed`, `busy`, `cancelled`, `workerCrashed`, `workerMissing`, `startFailed`, `protocolMismatch`. A worker that crashes fails only the calls it was running; the next call starts another.

On a server

By default the workers have the settings of the .NET library on your own machine: a page may open any host, read local files through a `file:` base URL and run without a time limit. A server that prints pages other people send should start its workers with SSRS2's server settings: a page then loads nothing but a URL's own host, private addresses and `file:` are refused, report code is always restricted, and a render stops at 2,000 pages or 120 seconds.

```
configure({ env: { SSRS2_WORKER_POLICY: 'service' } });

// changed as JSON over those settings: 500 pages at most, and pages may open
cdn.example.com
configure({
  env: {
    SSRS2_WORKER_POLICY: 'service',
    SSRS2_WORKER_SETTINGS: JSON.stringify({ maxPages: 500, network: { allowedHosts:
['cdn.example.com'] } }),
  },
});
```

Fonts

The engine brings faces drawn for Arial, Helvetica, Times New Roman, Courier New, Calibri, Cambria and Segoe UI (Latin, Greek, Cyrillic and Hebrew). Arabic, Chinese, Japanese, Korean, Thai, the Indic scripts and emoji are drawn with the machine's own fonts: macOS and Windows have them, a Linux container usually does not (`apt-get install -y fonts-noto-core fonts-noto-cjk fonts-noto-color-emoji`). The fonts guide

Licence

The SSRS2 licence covers Weblitz and SSRS2's reports. Give the key as the `SSRS2_LICENSE` environment variable or as `configure({ license: 'SSRS2-...' })`. Without one, PDF and image pages carry a trial watermark and a result's `trial` is true. Buy a licence