# Getting started

Weblitz prints HTML, CSS and the page's scripts to PDF from a .NET library, as Chrome prints them. Every picture of a page in this documentation is a page of a PDF Weblitz printed from the code shown beside it, made on 2026-11-23 on macOS; each PDF is a download.

## Install

Weblitz is the NuGet package `WeblitzPdf`, for .NET 8, 9, 10 and 11 on Windows, Linux and macOS. Version 2.0 is a release candidate: the latest is `2.0.0-rc.2`, a prerelease, so the command asks for prereleases.

```
dotnet add package WeblitzPdf --prerelease
```

Or in the project file:

```
<PackageReference Include="WeblitzPdf" Version="2.0.0-rc.2" />
```

Everything is in the package: the layout engine, the fonts it needs to print as Chrome does, and the script engine (V8, built in). Nothing to download at run time, no browser to start.

## From Node.js, Python, Java or any language

The same engine, at the same version, in a package of each language and in a Docker image: each carries the engine and the runtime it needs, so no .NET is installed. Their calls run in worker processes the package starts and keeps warm.

**.NET** `dotnet add package WeblitzPdf --prerelease` NuGet 2.0.0-rc.2

.NET 8, 9, 10 and 11 on Windows, Linux and macOS, x64 and ARM64

**Node.js** `npm install weblitzpdf` npm 2.0.0-rc.2

Node.js 22 or later, on Linux with glibc (x64, Arm64), macOS (Arm64, x64) and Windows (x64, Arm64)

**Python** `pip install --pre weblitz` PyPI 2.0.0rc2

Python 3.9 or later, on Linux with glibc (x64, Arm64), macOS (Arm64, x64) and Windows (x64, Arm64)

**Java** `com.weblitzpdf:weblitzpdf:2.0.0-rc.2` Maven 2.0.0-rc.2

Java 17 or later, on Linux with glibc (x64, Arm64), macOS (Arm64, x64) and Windows (x64, Arm64)

**HTTP**

`docker run -p 8080:8080 ghcr.io/antoinegou/weblitz-server:2.0.0-rc.2`

2.0.0-rc.2

Docker, linux/amd64 and linux/arm64

The Node.js, Python and Java packages' options are Puppeteer's and Playwright's `page.pdf()` options under their own names; the server takes Puppeteer's. Every package is version 2.0.0-rc.2, which Python spells 2.0.0rc2. Pictures of a page, from each: pictures.

## Your licence key

Weblitz is covered by the SSRS2 licence: one key, the one that starts with `SSRS2-`, for SSRS2's reports and for Weblitz. Activate it once when your application starts, or set the environment variable `SSRS2_LICENSE`, which is read at the first render.

```
using WeblitzPdf;

License.Activate("SSRS2-...");          // throws when the key is not valid
// or: License.TryActivate(key, out var error)
Console.WriteLine(License.LicensedTo);  // the company the key names
```

Without a key, Weblitz prints every page with a trial watermark, and does everything else the same: try it all before you buy. License in the reference lists what the key says (its company, expiry, the versions it covers).




Printed without a key. The PDF

# Your first PDFs

Four ways in: HTML text, a file with its folder, a web address, a stream. Each call is static and thread-safe, and returns an HtmlPdfResult: the PDF, its page count, and notes on what could not be done.

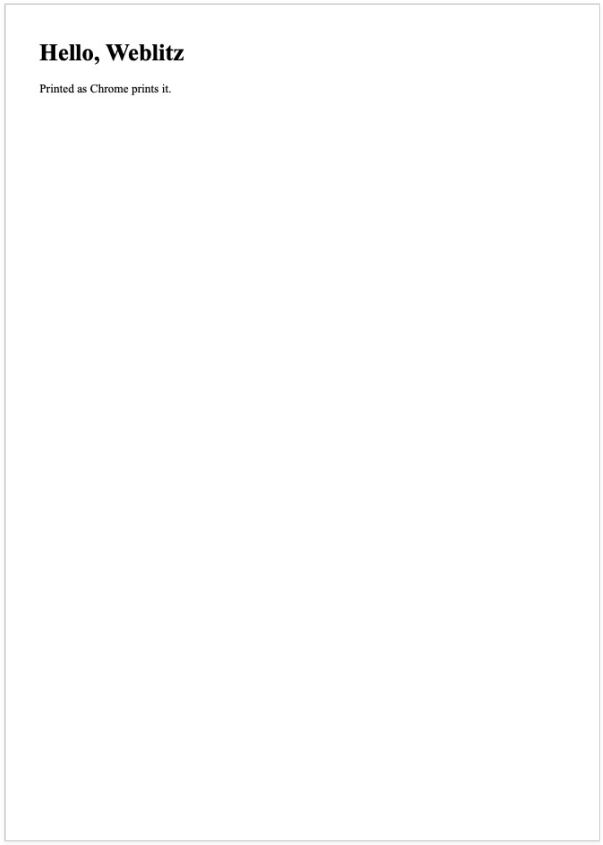
# Your first PDF: HTML text

The smallest program: HTML in, PDF bytes out.

Reference: `HtmlToPdf.Render`

### HTML text

```
var result = HtmlToPdf.Render("<h1>Hello, Weblitz</h1><p>Printed as Chrome prints it.
</p>");
File.WriteAllBytes("hello.pdf", result.Pdf!);
```




Pages 1
File 37,271 bytes (36.4 KB) · the PDF

# A file and its folder

RenderFile reads the file's stylesheet, pictures, fonts and scripts from its own folder, and nothing outside it.

Reference: `HtmlToPdf.RenderFile`

▶ The page printed: `letter/index.html`

### A folder: HTML, CSS, a logo

```
var result = HtmlToPdf.RenderFile(path);
```




Pages 1
File 50,768 bytes (49.6 KB) · the PDF

# A web address

RenderUrl fetches the page and what it loads. With the network left off, the page's own host is allowed (here a server the generator starts on this machine, as `assets`).

Reference: `HtmlToPdf.RenderUrl`, `HtmlPdfOptions.Network`

▶ The page printed: `letter/index.html`

### RenderUrl

```
var result = HtmlToPdf.RenderUrl(new Uri(assets, "letter/index.html"));
```




Pages 1
File 50,768 bytes (49.6 KB) · the PDF

## Into a stream

Render(html, stream) writes the PDF to a stream (the result's Pdf is then null): a file, a response, a blob.

Reference: `HtmlToPdf.Render`

### To a file stream

```
using var file = File.Create(output);
var result = HtmlToPdf.Render("<h1>Straight into a stream</h1>", file);
```

| | |
|---|---|
| Pages | 1 |
| File | 18,330 bytes (17.9 KB) · the PDF |
| Pdf | null: written to the stream |

## An ASP.NET Core endpoint

An endpoint returning a PDF made from your own HTML (a Razor view rendered to a string, a template, a page of your site). Render runs on the calling thread; RenderAsync runs it on the thread pool.

```
using WeblitzPdf;

app.MapGet("/invoices/{id}.pdf", async (int id, InvoiceViews views) =>
{
    var html = await views.Html(id);
    var result = await HtmlToPdf.RenderAsync(html, new HtmlPdfOptions {
Title = $"Invoice {id}" });
    return Results.File(result.Pdf!, "application/pdf", $"invoice-
{id}.pdf");
});
```

`InvoiceViews` stands for your own code that makes the invoice's HTML.

# Next

### Every option, rendered

Paper, margins, scale, backgrounds, pages, scripts, the network, compression, passwords, limits, notes, the browser a page sees.

### API reference

Every public type and member, from the engine's own documentation.

### Printing like Chrome

What happens between your HTML and the PDF.

### Fonts

The machine's fonts, the bundled faces, Linux containers.