Tutorials

(https://www.nuget.org/packages/IronPdf/)

Install with NuGet **nuget.org/packages/IronPdf** **(https://www.nuget.org/packages/IronPdf/)**

```
PM > Install-Package IronPdf
```

Explore the Docs

Get Started (/docs/)

Features (/features/)

Code Examples (/examples/using-html-to-create-a-pdf)

Tutorials

HTML to PDF (/tutorials/html-to-pdf/)

Editing PDFs in C# (/tutorials/csharp-edit-pdf-complete-tutorial/)

Debug HTML with Chrome (/tutorials/pixel-perfect-html-to-pdf/)

ASPX to PDF (/tutorials/aspx-to-pdf/)

VB.Net PDF (/tutorials/vb-net-pdf/)

.NET Core (/tutorials/dotnet-core-pdf-generating/)

How-Tos

Troubleshooting

Product Updates

API Reference (/object-reference/api/)

HTML to PDF C# Conversion

As the developers of IronPDF, we understand that PDF documents made by IronPDF not only need look perfect, but also need to look exactly how our customers expect them to. In this C# PDF tutorial will teach you how to build an HTML to PDF converter in your C# applications, projects, and websites. We will build a C# html-to-pdf converter. The output PDF documents from IronPDF are pixel identical to the PDF functionality in a Google Chrome web browser.

Steps to convert HTML to PDF in C#: First download the Library from Nuget, then choose your HTML document to convert and use the ChromePdfRenderer.RenderHtmlAsPdf Method to convert any HTML (HTML5) into a PDF. This approach works in both .NET6 and .NET Core.

Using the IronPDF C# Library we will:

- Create a PDF document from HTML string or HTML file as the 'content' within a C# application.
- Apply editing and PDF generation functionality in C#

- Convert URLs to PDF without losing any formatting

Overview

How to Convert HTML to PDF in C#

1. Download the HTML to PDF C# library
2. Convert to PDF from a HTML String in .NET C#
3. Convert Existing HTML URL to PDF
4. Convert Existing HTML Page(s) to PDF
5. Convert C# HTML to PDF using ChromePdfRenderer class
6. Convert PDFs by Applying HTML Templating
7. Convert to PDF including Attaching a Cover Page
8. Convert to PDF including a Watermark

Convert HTML to PDF in VB.NET and C# with IronPDF

Creating PDF files programmatically in .NET can be a frustrating task. The PDF document file format was designed more for printers than for developers. And C# doesn't have many suitable libraries or features for PDF generation built-in, many of the libraries that are on the market do not work out-of-the-box, and cause further frustration when they require multiple lines of code to accomplish a simple task.

The C# HTML to PDF conversion tool we will be using in this tutorial is IronPDF by Iron Software, a highly popular C# PDF generation and editing library. This library has comprehensive PDF editing and generation functionality, works completely out-of-the-box, does exactly what you need it to do in the least amount of lines, and has outstanding documentation of its 50+ features (https://ironpdf.com/features/). IronPDF stands out in that it supports **.NET 6 and .NET 5**, .NET Core, Standard, and Framework on Windows, macOS, Linux, Docker, Azure, and AWS.

With C# and IronPDF, the logic to "generate a PDF document (/blog/using-ironpdf/csharp-generate-pdf-tutorial/)" or "HTML to PDF conversion (/examples/using-html-to-create-a-pdf/)" is straightforward. Becuase of IronPDF's advanced Chrome Renderer, most or all of the PDF document design and layout will use existing HTML assets.

This method of dynamic PDF generation in .NET with HTML5 works equally well in console applications, windows forms applications, WPF, as well as websites and MVC.

IronPDF also supports debugging of your HTML with Chrome for Pixel Perfect PDFs. A tutorial for setting this up can be found here (/tutorials/pixel-perfect-html-to-pdf/).

VB.NET : Convert HTML to PDF

IronPDF is a C# Library that allows developers to create PDF documents easily in C#, F#, and VB.NET for .NET Core and .NET Framework. This ensures that we, as .NET coders, do not need to learn proprietary file formats or new APIs. We can easily output dynamic PDF files from our programs and web applications.

IronPDF Features:

- Generating PDFs from: HTML, URL, JavaScript, CSS and many image formats
- Adding headers/footers, signatures, attachments, and passwords and security
- Performance optimization: Full Multithreading and Async support

Step 1

1. Download the HTML to PDF .NET Library from IronPDF free from NuGet

Download DLL

Download DLL

Manually install into your project

or

Install with NuGet

```
PM > Install-Package IronPdf
```

nuget.org/packages/IronPdf/

Visual Studio - NuGet Package Manager

In Visual Studio, right click on your project solution explorer and select `Manage NuGet Packages...`, From there simply search for IronPDF and install the latest version to your solution... click OK to any dialog boxes that come up. This will also work just as well in VB.NET projects.

Visual Studio - Package Manager Console

Alternatively, in Visual Studio, navigate to the `tools` menu at the top and select `NuGet Package Manager` and choose `Package Manager Console` from the menu. In the shell that opens, paste the following and press enter:

```
PM > Install-Package IronPdf
```

IronPDF on NuGet Website

For a comprehensive rundown of IronPDF's features, compatibility and downloads, please check out IronPDF on NuGet's official website: (https://www.nuget.org/packages/IronPdf)https://www.nuget.org/packages/IronPdf (https://www.nuget.org/packages/IronPdf)

Install via DLL

Another option is to install the IronPDF DLL directly. IronPDF can be downloaded and manually installed to the project or GAC from (/packages/IronPdf.zip)https://ironpdf.com/packages/IronPdf.zip (https://ironpdf.com/packages/IronPdf.zip)

How to Tutorials

2. Create a PDF with an HTML String in C# .NET

How to: Convert HTML String to PDF? It is a very efficient and rewarding skill to *create a new PDF file in C#*.

We can simply use the ChromePdfRenderer.RenderHtmlAsPdf (https://ironpdf.com/object-reference/api/IronPdf.ChromePdfRenderer.html) method to turn any HTML (HTML5) string into a PDF. **C# HTML to PDF rendering** is undertaken by a fully functional version of the Google Chromium engine, embedded within IronPDF DLL.

```
using IronPdf;

var Renderer = new IronPdf.ChromePdfRenderer();
var PDF = Renderer.RenderHtmlAsPdf("<h1>Hello IronPdf</h1>");
PDF.SaveAs("pixel-perfect.pdf");
```

`RenderHtmlAsPdf` fully supports HTML5, CSS3, JavaScript, and Images. If these assets are on a hard disk, we may wish to set the second parameter of `RenderHtmlAsPdf` to the directory of the assets.

IronPDF will render your HTML exactly as it appears in Chrome

We have a full tutorial dedicated to allowing you to set up Chrome for full HTML debugging to make sure the changes you see there when editing your HTML, CSS, and JavaScript are pixel-perfect the same as the output PDF from IronPDF when you choose to render. Please find the tutorial here: How to Debug HTML in Chrome to Create Pixel Perfect PDFs (/tutorials/pixel-perfect-html-to-pdf/).

BaseUrlPath:

```
// this will render C:\MyProject\Assets\image1.png
var pdf = Renderer.RenderHtmlAsPdf("<img src='image1.png'/>", @"C:\MyProject\Assets\");
```

All referenced CSS stylesheets, images and JavaScript files will be relative to the `BaseUrlPath` and can be kept in a neat and logical structure. You may also, of course opt to reference images, stylesheets and assets online, including web-fonts such as Google Fonts (/docs/questions/webfonts-azure/) and even jQuery.

3. Export a PDF Using Existing URL

(URL to PDF)

Rendering existing URLs as PDFs with C# is very efficient and intuitive. This also allows teams to split PDF design and back-end PDF rendering work across multiple teams.

Lets render a page from Wikipedia.com in the following example:

```
// Create a PDF from any existing web page
var Renderer = new IronPdf.ChromePdfRenderer();
var pdf = Renderer.RenderUrlAsPdf("https://en.wikipedia.org/wiki/PDF");
pdf.SaveAs("wikipedia.pdf");
```

You will notice that hyperlinks and even HTML forms are preserved within the PDF generated by our C# code.

When rendering existing web pages we have some tricks we may wish to apply:

3.1. Print and Screen CSS

In modern CSS3 we have css directives for both print and screen. We can instruct IronPDF to render "Print" CSSs which are often simplified or overlooked. By default "Screen" CSS styles will be rendered, which IronPDF (https://ironpdf.com/) users have found most intuitive.

```
Renderer.RenderingOptions.CssMediaType = IronPdf.Rendering.PdfCssMediaType.Screen;
//or
Renderer.RenderingOptions.CssMediaType = IronPdf.Rendering.PdfCssMediaType.Print;
```

Main Page: A full comparison with images of Screen and Print can be found here (/tutorials/pixel-perfect-html-to-pdf/#decide-to-use-css-media-type-print-or-screen).

3.2. JavaScript

IronPDF supports JavaScript, jQuery and even AJAX. We may need to instruct IronPDF to **wait** for JS or ajax (/docs/questions/javascript-to-pdf/) to finish running before rendering a snapshot of our web-page.

```
Renderer.RenderingOptions.EnableJavaScript = true;
Renderer.RenderingOptions.RenderDelay = 500; //milliseconds
```

We can demonstrate compliance with the JavaScript standard by rendering an advanced d3.js JavaScript chord chart (https://bl.ocks.org/mbostock/4062006) from a CSV dataset like this:

```
// Create a PDF Chart a live rendered dataset using d3.js and javascript
var Renderer = new ChromePdfRenderer();
var pdf = Renderer.RenderUrlAsPdf("https://bl.ocks.org/mbostock/4062006");
pdf.SaveAs("chart.pdf");
```

3.3. Responsive CSS

HTML to PDF using response CSS in .NET! Responsive web pages (/docs/questions/html-to-pdf-responsive-css/) are designed to be viewed in a browser. IronPDF does not open a real browser window within your server's OS. This can lead to responsive elements rendering at their smallest size.

We recommend using **Print** css media types to navigate this issue. Print CSS should not normally be responsive.

```
Renderer.RenderingOptions.CssMediaType = ChromePdfRenderer.PdfCssMediaType.Print;
```

4. How To: Generate PDF From Existing HTML Page(s)

We can also render any HTML page to PDF on our hard disk. All relative assets such as CSS, images and js will be rendered as if the file had been opened using the **file://** protocol.

```
// Create a PDF from an existing HTML using C#
 var Renderer = new IronPdf.ChromePdfRenderer();
 var PDF = Renderer.RenderHtmlFileAsPdf("Assets/TestInvoice1.html");
 PDF.SaveAs("Invoice.pdf");
```

This method has the advantage of allowing the developer the opportunity to test the HTML content in a browser during development. We recommend Chrome as it is the web browser on which IronPDF's rendering engine is based.

To convert XML to PDF you can use XSLT templating to print your XML content to PDF (../../docs/questions/xml-to-pdf/).

5. Add Headers And Footers

Headers and footers can be added to PDFs when they are rendered, or to existing PDF files using IronPDF.

With IronPDF, Headers and footers can contain simple text based content using the *SimpleHeaderFooter* class - or with images and rich html content using the *HtmlHeaderFooter* class.

```
// Create a PDF from an existing HTML
var Renderer = new IronPdf.ChromePdfRenderer();

Renderer.RenderingOptions.MarginTop = 50;  //millimeters
Renderer.RenderingOptions.MarginBottom = 50;
Renderer.RenderingOptions.CssMediaType = ChromePdfRenderer.PdfCssMediaType.Print;

Renderer.RenderingOptions.TextHeader = new TextHeaderFooter()
{
    CenterText = "{pdf-title}",
    DrawDividerLine = true,
    FontSize = 16
};

Renderer.RenderingOptions.TextFooter = new TextHeaderFooter()
{
    LeftText = "{date} {time}",
    RightText = "Page {page} of {total-pages}",
    DrawDividerLine = true,
    FontSize = 14
};

var pdf = Renderer.RenderHtmlFileAsPdf("assets/TestInvoice1.html");
pdf.SaveAs("Invoice.pdf");

// This neat trick opens our PDF file so we can see the result
System.Diagnostics.Process.Start("Invoice.pdf");
```

5.1. HTML Headers and Footers

The HtmlHeaderFooter class allows for rich headers and footers to be generated using HTML5 content which may even include images, stylesheets and hyperlinks.

```
Renderer.RenderingOptions.HtmlFooter = new HtmlHeaderFooter()
{ HtmlFragment = "<div style='text-align:right'><em style='color:pink'>page {page} of {total-pages}</em></div>" };
```

5.2. Dynamic Data in PDF Headers and Footers

We may "mail-merge" content into the text and even HTML of headers and footers using placeholders such as:

- {page} for the current page number
- {total-pages} for the total number of pages in the PDF
- {url} for the URL of the rendered PDF if rendered from a web page
- {date} for today's date
- {time} for the current time
- {html-title} for the *title* attribute of the rendered HTML document
- {pdf-title} for the document title, which may be set via ChromePdfRenderOptions

6. C# HTML to PDF Conversion Settings

There are many nuances to how our users and clients may expect PDF content to be rendered.
The `ChromePdfRenderer` class contains a **RenderingOptions** property which can be used to set these options.

For example we may wish to choose to only accept " `print` " style CSS3 directives:

```
Renderer.RenderingOptions.CssMediaType = IronPdf.Rendering.PdfCssMediaType.Print;
```

We may also wish to change the size of our print margins to create more whitespace on the page, to make room for large headers or footers, or even set zero margins for commercial printing of brochures or posters:

```
Renderer.RenderingOptions.MarginTop = 50;  //millimeters
Renderer.RenderingOptions.MarginBottom = 50;
```

We may wish to turn on or off background images from HTML elements:

```
Renderer.RenderingOptions.PrintHtmlBackgrounds = true;
```

It is also possible to set our output PDFs to be rendered on any virtual paper size - including portrait and landscape sizes and even custom sizes which may be set in millimeters or inches.

```
Renderer.RenderingOptions.PaperSize = ChromePdfRenderer.PdfPaperSize.A4;
Renderer.RenderingOptions.PaperOrientation = IronPdf.Rendering.PdfPaperOrientation.Landscape;
```

Full documentation of the HTML C# PDF Creator (/use-case/csharp-pdf-creator/) Settings may be found at https://ironpdf.com/object-reference/api/IronPdf.ChromePdfRenderer.html (https://ironpdf.com/object-reference/api/IronPdf.ChromePdfRenderer.html)

The full set of PDF Print Options includes:

- **CreatePdfFormsFromHtml** Turns all HTML forms elements into editable PDF forms.
- **CssMediaType** `Screen` or `Print` CSS Styles and StyleSheets. See our full in-depth tutorial with comparison images (https://ironpdf.com/tutorials/pixel-perfect-html-to-pdf/).
- **CustomCssUrl** Allows a custom CSS style-sheet to be applied to HTML before rendering. May be a local file path, or a remote URL.
- **DPI** Printing output Dots Per Inch (DPI). 300 is standard for most print jobs. Higher resolutions produce clearer images and text, but also larger PDF files.
- **EnableJavaScript** Enables JavaScript and JSON to be executed before the page is rendered. Ideal for printing from Ajax / Angular Applications. Also see RenderDelay (https://ironpdf.com/troubleshooting/render-delay-timeout/).
- **FirstPageNumber** First page number to be used in PDF headers and footers.
- **FitToPaper** Where possible, zooms the PDF content to 1 page width.
- **TextFooter** Sets the footer content for every PDF page as a String. Supports 'mail-merge'
- **TextHeader** Sets the header content for every PDF page as a String. Supports 'mail-merge'
- **HtmlFooter** Sets the footer content for every PDF page as HTML
- **HtmlHeader** Sets the header content for every PDF page as HTML
- **MarginBottom** Paper margin in millimeters. Set to zero for border-less and commercial printing applications
- **MarginLeft** Paper margin in millimeters
- **MarginRight** Paper margin in millimeters
- **MarginTop** Paper margin in millimeters. Set to zero for border-less and commercial printing applications

- **PdfPaperOrientation** Paper orientation for new document. Full explanation and accompanying code example (https://ironpdf.com/examples/pdf-page-orientation/).
- **PageRotation** Page rotation from existing document. Full explanation and accompanying code example (https://ironpdf.com/examples/pdf-page-orientation/).
- **PaperSize** Set an output paper size for PDF pages. `System.Drawing.Printing.PaperKind`. Use `SetCustomPaperSize(int width, int height)` for custom sizes.
- **PrintHtmlBackgrounds** Prints background-colors and images from HTML.
- **RenderDelay** Milliseconds delay to wait after HTML is rendered before printing. This can use useful when considering the rendering of JavaScript, Ajax or animations.
- **Title** PDF Document Name and Title meta-data. Not required.
- **Zoom** The zoom level in %. Enlarges the rendering size of HTML documents.

7. Apply HTML Templating

To template or "batch create" PDFs is a common requirement for Internet and website developers.

Rather than templating a PDF document itself, with IronPDF we can template our HTML using existing, well tried technologies. When the HTML template is combined with data from a query-string or database we end up with a dynamically generated PDF document.

In the simplest instance, using the C# String.Format method is effective for basic "mail-merge"

```
/**
HTML Templating
anchor-apply-html-templating
**/
String.Format("<h1>Hello {0} !</h1>","World");
```

If the HTML file is longer, often we can use arbitrary placeholders such as `[[NAME]]` and replace them with real data later.

The following example will create 3 PDFs, each personalized to a user.

```
var HtmlTemplate = "<p>[[NAME]]</p>";

var Names = new[] { "John", "James", "Jenny" };

foreach (var name in Names) {
    var HtmlInstance = HtmlTemplate.Replace("[[NAME]]", name);
    var Pdf = Renderer.RenderHtmlAsPdf(HtmlInstance);
    Pdf.SaveAs(name + ".pdf");
}
```

7.1. Advanced Templating With Handlebars.NET

A sophisticated method to merge C# data with HTML for PDF generation is using the Handlebars Templating standard.

Handlebars makes it possible to create dynamic HTML from C# objects and class instances including database records. Handlebars is particularly effective where a query may return an unknown number of rows such as in the generation of an invoice.

We must first add an additional NuGet Package to our project: https://www.nuget.org/packages/Handlebars.NET/ (https://www.nuget.org/packages/Handlebars.NET/)

```
var source =
  @"<div class=""entry"">
    <h1>{{title}}</h1>
    <div class=""body"">
      {{body}}
    </div>
  </div>";

var template = Handlebars.Compile(source);

var data = new {
    title = "My new post",
    body = "This is my first post!"
};

var result = template(data);

/* Would render:
<div class="entry">
  <h1>My New Post</h1>
  <div class="body">
    This is my first post!
  </div>
</div>
*/
```

To render this HTML we can simply use the `RenderHtmlAsPdf` method.

```
var Renderer = new IronPdf.ChromePdfRenderer();
var mypdf = Renderer.RenderHtmlAsPdf(HtmlInstance);
mypdf.SaveAs("Handlebars.pdf")
```

You can learn more about the handlebars html templating standard and its C# using from https://github.com/rexm/Handlebars.NET (https://github.com/rexm/Handlebars.NET)

7.2. Add Page Breaks using HTML5

A common requirement in a PDF document is for pagination. Developers need to control where PDF pages start and end for a clean, readable layout.

The easiest way to do this is with a less known CSS trick which will render a page break into any printed HTML document.

The provided HTML works, but is hardly best practice. We found this example to be very helpful in our understanding of a neat and tidy way to lay out multi-page html content.

The How-Tos outline more tips and tricks with Page Breaks (/docs/questions/html-to-pdf-page-breaks/)

8. Attach a Cover Page to a PDF

IronPDF makes it easy to Merge PDF documents. The most common usage of this technique is to add a cover page or back page to an existing rendered PDF document.

To do so, we first render a cover page, and then use the `PdfDocument.Merge()` static method to combine the 2 documents.

```
var PDF = Renderer.RenderUrlAsPdf("https://www.nuget.org/packages/IronPdf/");
PdfDocument.Merge(new PdfDocument("CoverPage.pdf"), PDF).SaveAs("Combined.Pdf");
```

A full code example can be found here: PDF Cover Page Code Example (https://ironpdf.com/examples/pdf-cover-page/)

9. Add a Watermark

A final C# PDF (/use-case/csharp-pdf/) feature that IronPDF supports is to add a watermark to documents. This can be used to add a notice to each page that a document is "confidential" or a "sample".

```
var Renderer = new ChromePdfRenderer();
var myPdf =  Renderer.RenderUrlAsPdf("https://www.nuget.org/packages/IronPdf");

// Watermarks all pages with red "SAMPLE" text at a custom location.
// Also adding a link to the watermark on-click
myPdf.WatermarkAllPages("<h2 style='color:red'>SAMPLE</h2>", IronPdf.Editing.WaterMarkLocation.MiddleCenter, 50, -45, "https://www.nuget.or
g/packages/IronPdf");

pdf.SaveAs(@"C:\Path\To\Watermarked.pdf");
```

A full code example can be found here: PDF Watermarking Code Example (https://ironpdf.com/examples/pdf-watermarking/)

10. Download C# Source Code

The full **free HTML to PDF converter C# Source Code** for this tutorial is available to download as a zipped Visual Studio 2017 project file. It will use its rendering engine to generate PDF document objects in C#.

Download this tutorial as a Visual Studio project (downloads/CSharp-Html-To-Pdf-Tutorial.zip)

The free download contains everything you need to create a PDF from HTML - including working C# PDF code examples code for:

1. Convert an HTML String to PDF using C#
2. HTML file to PDF in C# (supporting CSS, JavaScript and images)
3. C# HTML to PDF using a URL ("url to pdf")
4. C# PDF editing and settings examples
5. Rendering JavaScript canvas charts such as d3.js to a PDF
6. The PDF Library for C#

Class Reference

Developers may also be interested in the IronPdf.PdfDocument Class reference:

https://ironpdf.com/object-reference/api/IronPdf.PdfDocument.html (https://ironpdf.com/object-reference/api/IronPdf.PdfDocument.html)

This object model shows how PDF documents may be:

- Encrypted and password protected
- Edited or 'stamped' with new html content
- Enhanced with foreground and background images
- Merged, joined, truncated and spliced at a page or document level
- OCR processed to extract plain text and images

11. Blazor HTML to PDF

Adding HTML to PDF functionality to your Blazor server is easy, simply:

1. Create a new Blazor server project or use an existing one
2. Add the IronPDF library to your project using NuGet
3. Add a new Razor Component or use an existing one
4. Add a InputTextArea and link it to IronPDF
5. Let IronPDF take care of the rest and deploy

The full step-by-step guide with pictures and code examples can be found here (https://ironpdf.com/docs/questions/blazor-tutorial/).

12. Compare with Other PDF Libraries

PDFSharp

PDFSharp is a free open source library which allows logical editing and creation of PDF documents in .NET.

A key difference between PDFSharp and IronPDF is that IronPDF has an embedded Web Browser which allows faithful creation of PDFs from HTML, CSS, JS and images.

The IronPDF API also differs from PDFSharp in that it is based around use cases rather than the technical structure of PDF documents. Many find this more logical and intuitive to use.

It can convert HTML to PDF, but HTML to PDF conversion is limited: including .html files to PDF files.

WKHtmlToPdf

WKHtmlToPdf is a free, open source library written in C++ which allows PDF documents to be rendered from HTML.

A key difference between WKHtmlToPdf and IronPDF is that IronPDF is written in C# and is stable and thread safe for use in .NET applications and Websites.

IronPDF also fully supports CSS3 and HTML5, where as WKHtmlToPdf is almost a decade out of date.

The IronPDF API also differs from WKHtmlToPdf in that it has a large and advanced API allowing PDF documents to be edited, manipulated, compressed, imported, exported, signed, secured and watermarked.

HTML to PDF conversion with WKHtmlToPdf is stable but utilizes a very outdated rendering engine.

iTextSharp

iTextSharp is an open source partial port of the iText java library for PDF generation and editing. Convert HTML to PDF- that is possible, but I notice its rendering was limited to what is available in Java or uses HTML to PDF conversion from WKHtmlToPdf under the LGPL open sourced license.

A key difference with HTML to PDF between C# iTextSharp (/use-case/csharp-itextsharp/) and IronPDF is that IronPDF has more advanced and accurate HTML-To-PDF rendering by using an embedded Chrome based web browser rather than the legacy WKHtmlToPdf used in iText.

The IronPDF API also differs from iTextSharp in that IronPDF has explicit licenses for commercial or private usage, where as iTextSharp's AGLP license is only suitable for applications where the full source code is presented for free to every user - even users across the internet.

A full breakdown of the differences is available in our iTextSharp (/docs/questions/itextsharp/) C# documentation page.

Other Commercial Libraries

Aspose PDF, *Spire PDF*, *EO PDF*, and *SelectPdf* are competitor .NET commercial PDF libraries by other vendors. IronPDF has a comparatively strong feature set, excellent compatibility, well-written documentation, and a fair price point. You can see the comparison between IronPDF, competitors, and Chrome itself here (https://ironpdf.com/tutorials/pixel-perfect-html-to-pdf/#what-is-ironpdf-s-chrome-renderer).

13. Watch HTML to PDF Tutorial Video

Tutorial Quick Access

Download this Tutorial as C# Source Code

The full free HTML to PDF C# Source Code for this tutorial is available to download as a zipped Visual Studio project file.

Download (downloads/CSharp-Html-To-Pdf-Tutorial.zip)

Explore this Tutorial on GitHub

The source code for this project is available in C# and VB.NET on GitHub.

Use this code as an easy way to get up and running in just a few minutes. The project is saved as a Microsoft Visual Studio 2017 project, but is compatible with any .NET IDE.

C# HTML to PDF > (https://github.com/iron-software/c-sharp-html-to-pdf-tutorial)
VB.NET HTML to PDF > (https://github.com/iron-software/vb.net-html-to-pdf-tutorial)

Download the C# PDF Quickstart guide

To make developing PDFs in your .NET applications easier, we have compiled a quick-start guide as a PDF document. This "Cheat-Sheet" provide quick access to common functions and examples for generating and editing PDFs in C# and VB.NET - and may help save time in getting started using IronPDF in your .NET project.

Download (/csharp-pdf.pdf)

View the API Reference

Explore the API Reference for IronPDF, outlining the details of all of IronPDF’s features, namespaces, classes, methods fields and enums.

View the API Reference > (/object-reference/api/IronPdf.html)

Jean Ashberg

.Net Software Engineer

Jean is an independent software developer for corporate internal information solutions based in Massachusetts, USA.

Jean was an early adopter of IronPDF, and has repeatedly been involved in ‘speccing-out’ product improvement and building a roadmap to creating a single stable library for C# that covers all major PDF product feature use cases.

Next Tutorial (/tutorials/c