IRONPDF

Tutorial

How to create PDFs in C# with HTML to PDF Converter

HTML to PDF

```
1. // Render any HTML fragment or document to HTML
2. var Renderer = new IronPdf.HtmlToPdf();
3. var PDF = Renderer.RenderHtmlAsPdf("<h1>Hello IronPdf</h1>");
4. var OutputPath = "HtmlToPDF.pdf";
5. PDF.SaveAs(OutputPath);
6. // This neat trick opens our PDF file so we can see the result
   in our default PDF viewer
7. System.Diagnostics.Process.Start(OutputPath);
```

IRONPDF

HTML to PDF C# Conversion

by Jean Ashberg

Interact with the tutorial: https://ironpdf.com/tutorials/html-to-pdf/

Share the tutorial:

This C# PDF tutorial will guide you step-by-step how to convert an HTML page to PDF in ASP.NET C# applications and websites (C# htmltopdfconverter). With C# we can create PDF documents using HTML as the 'content' for the PDF, applying editing and generation functionality.

Table of Contents

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.

The tool we will be using in this tutorial is IronPDF, a popular C# PDF generation and editing library. This library has comprehensive PDF editing and generation functionality via HTML to PDF. IronPDF stands out in that it supports .NET Framework and **.NET Core** on Windows, Linux, Azure and MacOS.

With C# and IronPDF, creating PDF documents can be straightforward. Much of the PDF document design and layout can use existing HTML assets or be delegated to web design staff.

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 is compatible with any .Net Framework project from Version 4 upwards, .Net Core from version 2 upwards.

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:

- Human support directly from our .NET development team
- Works with the documents you already have, including HTML, ASPX forms, MVC views and image files
- Rapid installation with Microsoft Visual Studio
- FREE for development. Licenses from $399.

Step 1

1. Download the HTML to PDF .NET Library from IronPDF FREE

Download DLL
Manually install into your project

or

Install with NuGet
nuget.org/packages/IronPdf/

Install via NuGet

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... click ok to any dialog boxes that come up.

This will work in any C# .Net Framework project from Framework 4 and above, or .Net Core 2 and above. It will also work just as well in VB.Net projects.

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

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

Install via DLL

Alternatively, the IronPDF DLL can be downloaded and manually installed to the project or GAC from https://ironpdf.com/packages/IronPdf.zip

Remember to add this statement to the top of any **cs** class file using IronPDF:

```
using IronPdf;
```


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

C# HTML String to PDF is a very efficient and rewarding way to create a new PDF file in C#.

We can simply use the HtmlToPdf.RenderHtmlAsPdf 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.

```
1. // Render any HTML fragment or document to HTML
2. var Renderer = new IronPdf.HtmlToPdf();
3. var PDF = Renderer.RenderHtmlAsPdf("<h1>Hello IronPdf</h1>");
4. var OutputPath = "HtmlToPDF.pdf";
5. PDF.SaveAs(OutputPath);
6. // This neat trick opens our PDF file so we can see the result in our default PDF viewer
7. System.Diagnostics.Process.Start(OutputPath);
```

RenderHtmlAsPdf fully supports CSS, Javascript and Images. If these assets are on a hard disk, we may wish to set the second parameter of RenderHtmlAsPdf

BaseUrlPath:

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

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 and even jQuery.

3. Export a PDF Using Existing HTML URL

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:

```
1. // Create a PDF from any existing web page
2. var Renderer = new IronPdf.HtmlToPdf();
3. var PDF = Renderer.RenderUrlAsPdf("https://en.wikipedia.org/wiki/Portable_Document_Format");
4. PDF.SaveAs("wikipedia.pdf");
5. // This neat trick opens our PDF file so we can see the result
6. System.Diagnostics.Process.Start("wikipedia.pdf");
```

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 users have found most intuitive.

```
1. Renderer.PrintOptions.CssMediaType = PdfPrintOptions.PdfCssMediaType.Print;
2. //or
3. Renderer.PrintOptions.CssMediaType = PdfPrintOptions.PdfCssMediaType.Screen;
```

3.2. Javascript

IronPDF supports Javascript, jQuery and even AJAX. We may need to instruct IronPDF to **wait** for JS or ajax to finish running before rendering a snapshot of our web-page.

```
1. Renderer.PrintOptions.EnableJavaScript = true;
2. Renderer.PrintOptions.RenderDelay = 500; //milliseconds
```

We can demonstrate compliance with the Javascript standard by rendering an advanced d3.js Javascript chord chart from a csv dataset like this:

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

3.3. Responsive CSS

Responsive web pages 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.

```
1. Renderer.PrintOptions.CssMediaType = PdfPrintOptions.PdfCssMediaType.Print;
```

4. Generate a PDF From an Existing HTML File

We can also render any HTML file 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.

```
1. // Create a PDF from an existing HTML using C#
2.  var Renderer = new IronPdf.HtmlToPdf();
3.  var PDF = Renderer.RenderHTMLFileAsPdf("Assets/TestInvoice1.html");
4.  var OutputPath = "Invoice.pdf";
5.  PDF.SaveAs(OutputPath);
```

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.

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.HtmlToPdf();
Renderer.PrintOptions.MarginTop = 50;  //millimeters
Renderer.PrintOptions.MarginBottom = 50;
Renderer.PrintOptions.CssMediaType = PdfPrintOptions.PdfCssMediaType.Print;
Renderer.PrintOptions.Header = new SimpleHeaderFooter()
{
    CenterText = "{pdf-title}",
    DrawDividerLine = true,
FontSize = 16
};
Renderer.PrintOptions.Footer = new SimpleHeaderFooter()
{
    LeftText = "{date} {time}",
    RightText = "Page {page} of {total-pages}",
    DrawDividerLine = true,
    FontSize = 14
};
var PDF = Renderer.RenderHTMLFileAsPdf("Assets/TestInvoice1.html");
var OutputPath = "Invoice.pdf";
PDF.SaveAs(OutputPath);
// This neat trick opens our PDF file so we can see the result
System.Diagnostics.Process.Start(OutputPath);
```

```
Renderer.PrintOptions.Footer = new SimpleHeaderFooter()
{
    LeftText = "{date} {time}",
    RightText = "Page {page} of {total-pages}",
    DrawDividerLine = true,
    FontSize = 14
};
```

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.PrintOptions.Footer = 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 the PrintOptions

6. C# HTML to PDF Settings

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

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

```
1. Renderer.PrintOptions.CssMediaType = PdfPrintOptions.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:

```
1. Renderer.PrintOptions.MarginTop = 50;  //millimeters
2. Renderer.PrintOptions.MarginBottom = 50;
```

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

```
1. Renderer.PrintOptions.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.

```
1. Renderer.PrintOptions.PaperSize = PdfPrintOptions.PdfPaperSize.A4;
2. Renderer.PrintOptions.PaperOrientation = PdfPrintOptions.PdfPaperOrientation.Landscape;
```

Full documentation of the HTML C# PDF Creator Settings may be found at
https://ironpdf.com/c%23-pdf-documentation/html/T_IronPdf_PdfPrintOptions.htm

The full set of PDF PrintOptions includes:

- **CreatePdfFormsFromHtml** Turns all HTML forms elements into editable PDF forms.
- **CssMediaType** Enables Media="screen" or "print" CSS Styles and StyleSheets.
- **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 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.
- **FirstPageNumber** First page number to be used in PDF headers and footers.
- **FitToPaperWidth** Where possible, fits the PDF content to 1 page width.
- **Footer** Sets the header content for every PDF page as Html or a String. Supports 'mail-merge'
- **GrayScale** Outputs a black-and-white PDFI documents
- **Header** Sets the footer content for every PDF page as Html or String. Supports 'mail-merge'
- **InputEncoding** The input character encoding as a string
- **JpegQuality** Quality of any image that must be re-sampled. 0-100
- **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
- **PaperOrientation** The PDF paper 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 Htm

---

7. Apply HTML Templating

To template or "batch create" PDFs is a common requirement for Intranet 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"

```
1.  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/

```
string 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.

```
IronPdf.HtmlToPdf Renderer = new IronPdf.HtmlToPdf();
Renderer.RenderHtmlAsPdf(HtmlInstance).SaveAs("Handelbars.pdf")
```

You can learn more about the handlebars html templating standard and its C# using from 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.

```
<div style='page-break-after: always;'> </div>
```

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.

```
<!DOCTYPE html>
<!--https://stackoverflow.com/questions/1630819/google-chrome-printing-page-breaks-->
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
    <title>Paginated HTML</title>
    <style type="text/css" media="print">
      div.page
      {
        page-break-after: always;
        page-break-inside: avoid;
      }
    </style>
  </head>
  <body>
    <div class="page">
      <h1>This is Page 1</h1>
    </div>
    <div class="page">
      <h1>This is Page 2</h1>
    </div>
    <div class="page">
      <h1>This is Page 3</h1>
    </div>
  </body>
</html>
```

The FAQ outlines more tips and tricks with 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.

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

9. Add a Watermark

A final C# PDF trick is to add a watermark to PDF documents. This can be used to add a notice to each page that a document is "confidential" or a "sample".

```
1. // Stamps a watermark onto a new or existing PDF
2. IronPdf.HtmlToPdf Renderer = new IronPdf.HtmlToPdf();
3. var pdf =  Renderer.RenderUrlAsPdf("https://www.nuget.org/packages/IronPdf");
4. pdf.WatermarkAllPages("<h2 style='color:red'>SAMPLE</h2>",
   PdfDocument.WaterMarkLocation.MiddleCenter, 50, -45, "https://www.nuget.org/packages/IronPdf");
5. pdf.SaveAs(@"C:\Path\To\Watermarked.pdf");
```

10. Download 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 2017 project file.

Download this tutorial as a Visual Studio project

The free download contains working C# PDF code examples code for:

1. Html Strings to PDFs using C#
2. Html files (supporting CSS, Javascript and images) to PDF
3. C# HTML to PDF using a URL
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/c%23-pdf-documentation/html/T_IronPdf_PdfDocument.htm

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. 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.

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.

The IronPDF API also differs from WKHtmlToPdf in that it has a large and advanced API allowing PDF documents to be edited, Manipulated Imported, Exported, Signed, Secured and Watermarked.

iTextSharp

iTextSharp is an open source partial port of the iText java library for PDF generation and editing.

A key difference with HTML to PDF between C# iTextSharp and IronPDF is that IronPDF is has more advanced and accurate HTML-To-PDF rendering by using an embedded Chrome based web browser.

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 C# FAQ.

Other Commercial Libraries

Aspose PDF, Spire PDF, EO PDF, and SelectPdf are competitor .Net commercial PDF libraries by other vendors. It is unfair for tutorials on this website to comment on them comparatively, as we clearly believe in the quality of IronPDF. Suffice to say: we believe IronPDF to have a comparatively strong feature set, excellent .Net Core compatibility and a fair price point.

Tutorial Quick Access

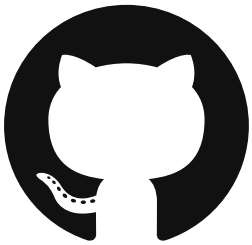
Download this Tutorial as C# Source Code

The full free HTML to PDF C# Source Code for this tutorial is available to download as a zipped Visual Studio 2017 project file.

Download

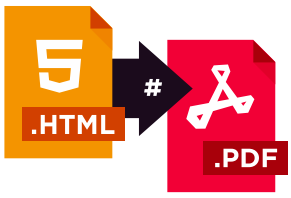
Explore this Tutorial on GitHub

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 >

VB.NET HTML to PDF >



Download 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

View the object reference

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

View the Object Reference >

The C# PDF solution you've been looking for.

Support

Open a support ticket
with our development team.

Ask a Question

Documentation

View code examples
and tutorials.

Get Started

Licensing

Free for development.
License from $399.

See Licenses

Try IronPDF Free

Get set up in 5 minutes.

Download