



## Tutorial

# How to create PDF files in VB .NET PDF Generator

### VB .NET PDF

```
1. Module Module1
2.     Sub Main()
3.         Dim renderer = New IronPdf.HtmlToPdf()
4.         Dim document = renderer.RenderHtmlAsPdf("<h1> My
5. First PDF in VB.Net</h1>")
6.         document.SaveAs("MyFirst.pdf")
7.     End Sub
8. End Module
```

# VB.NET PDF Generator

by Veronica Sillar

Interact with the tutorial: <https://ironpdf.com/tutorials/vb-net-pdf/>

Share the tutorial: [✉](#) [f](#) [@](#) [in](#) [t](#)

This tutorial will guide you step-by-step how to create and edit PDF files in VB.Net. This technique is equally valid for use in **ASP.NET web apps** as well as **console applications**, **Windows Services**, and **desktop programs**. We will use VB.NET to create PDF projects targeting .NET Framework 4 or .NET Core 2. All you need is a Visual Basic .NET development environment, such as Microsoft Visual Studio Community.

## Table of Contents

1. **Download the VB .NET PDF Library FREE from IronPDF**
2. **Create A PDF with VB.NET**
3. **Apply Styling to VB.Net PDF**
4. **Create PDF w/ Dynamic Content : 2 Methods**
5. **Edit PDF Files with VB.Net**
6. **More .NET PDF Tutorials**

# VB .NET Codes for PDF Creating and Editing with IronPDF

Render HTML to PDF with VB.NET, apply styling, utilize dynamic content, and edit your files easily. Creating PDFs is straightforward and compatible with .NET Framework 4 or .NET Core 2. And no need for proprietary file formats or pulling different API's.


This tutorial provides the documentation to walk you through each task step-by-step, all using the free for development [IronPDF software favored by developers](#). VB.NET code examples are specific to your use cases so you can see the steps easily in a familiar environment. This VB dot NET PDF Library has comprehensive creation and settings capabilities for every project, whether in ASP.NET applications, console, or desktop.

## Included with IronPDF:

- Ticket support direct from our .NET PDF Library development team (real humans!)
- Works with HTML, ASPX forms, MVC views, images, and all the document formats you already use
- Microsoft Visual Studio installation gets you up and running fast
- Unlimited free development, and licenses to go live starting at \$399

## Step 1

# 1. Download the VB .NET PDF Library FREE from IronPDF



**Download DLL**  
Manually install into your project

or



**Install with NuGet**  
[nuget.org/packages/IronPdf/](https://www.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;
```

## How to Tutorials

## 2. Create A PDF with VB.NET

Using **Visual Basic ASP.Net** to create a PDF file for the first time is surprising easy using IronPDF, as compared to libraries with proprietary design API's such as *iTextSharp*.

We can use HTML (with a pixel perfect rendering engine based on Google Chromium) to define the content of our PDF and simply render it to a file.

Here is our simplest code to create a PDF in VB.Net:

```
1.
2.  Module Module1
3.
4.     Sub Main()
5.         Dim renderer = New IronPdf.HtmlToPdf()
6.         Dim document = renderer.RenderHtmlAsPdf("<h1> My First PDF in VB.Net</h1>")
7.         document.SaveAs("MyFirst.pdf")
8.     End Sub
9.
10. End Module
```

This will produce a .NET generated PDF file containing your exact text, albeit lacking some design at this point.

We can improve upon this code by adding the header line **Imports IronPdf**. By adding the last line of code *System.Diagnostics.Process.Start*, we open the PDF in the operating system's default PDF viewer to make the project more meaningful.

```

1. Imports IronPdf
2.
3. Module Module1
4.
5.     Sub Main()
6.         Dim renderer = New HtmlToPdf()
7.         Dim document = renderer.RenderHtmlAsPdf("<h1> My First PDF in VB.Net</h1>")
8.         document.SaveAs("MyFirst.pdf")
9.         System.Diagnostics.Process.Start("MyFirst.pdf")
10.    End Sub
11.
12. End Module

```

An alternative method would be to render any existing web page from a URL to a PDF by using the elegant “RenderUrlAsPdf” method from IronPDF.

```

1. Imports IronPdf
2.
3. Module Module1
4.
5.     Sub Main()
6.         Dim renderer = New HtmlToPdf()
7.         Dim document = renderer.RenderUrlAsPdf("https://www.nuget.org/packages/IronPdf/")
8.         document.SaveAs("UrlToPdf.pdf")
9.         System.Diagnostics.Process.Start("UrlToPdf.pdf")
10.    End Sub
11.
12. End Module

```

If you’d like to generate your PDF in [PDF/A format](#), you’ll need to render in IronPDF first, then use Ghostscript to convert to PDF/A.

---

### 3. Apply Styling to VB.Net PDF

To style our PDF content in VB.Net, we can make full use of CSS, Javascript and images. We may link to local assets, or even to remote or CDN based assets such as Google Fonts. We can even use [DataURIs to embed images and assets as a string into your HTML](#).

For advanced design, we can use a 2 stage process:

1. First we develop and design our HTML perfectly. This task may involve in-house design staff, splitting the work load.
2. Render that file as a PDF using VB.Net and our PDF Library

## The VB.Net Code to render the HTML file as a PDF:

This method renders an HTML document as if it were opened as a file (*file:// protocol*).

```
1. Imports IronPdf
2.
3. Module Module1
4.
5.     Sub Main()
6.         Dim renderer = New HtmlToPdf()
7.         renderer.PrintOptions.CssMediaType = PdfPrintOptions.PdfCssMediaType.Print
8.         renderer.PrintOptions.PrintHtmlBackgrounds = False
9.         renderer.PrintOptions.PaperOrientation = PdfPrintOptions.PdfPaperOrientation.Landscape
10.        renderer.PrintOptions.RenderDelay = 150
11.
12.        Dim document = renderer.RenderHTMLFileAsPdf("C:\Users\jacob\Dropbox\Visual
13.        Studio\Tutorials\VB.Net.Pdf.Tutorial\VB.Net.Pdf.Tutorial\slideshow\index.html")
14.        document.SaveAs("Html5.pdf")
15.        System.Diagnostics.Process.Start("Html5.pdf")
16.    End Sub
17. End Module
```

We might also shorten that URL by adding a project relative file path such as:

```
1. Dim document = renderer.RenderHTMLFileAsPdf("../..\\slideshow\\index.html")
```

You can see that the *HtmlToPdf* renderer has a **PrintOptions** object which we can use in this example to:

- Set the CSS media type to 'print' so we see no screen-only CSS3 styles
- Ignore HTML backgrounds
- Set the PDF's virtual paper to Landscape orientation
- Add a small delay in rendering for the Javascript to finish processing

Our example HTML File uses Javascript, CSS3 and images. This HTML creates a dynamic, mobile-aware slideshow and was found at <https://github.com/leemark/better-simple-slideshow>

```
1. <!DOCTYPE html>
2. <html>
3.     <head>
4.         <meta charset="utf-8">
5.         <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.         <title>A simple DIY responsive slideshow made with HTML5, CSS3, and JavaScript</title>
7.         <meta name="description" content="">
8.         <meta name="viewport" content="width=device-width, initial-scale=1">
9.         <link href='http://fonts.googleapis.com/css?family=Open+Sans|Open+Sans+Condensed:700'
10.        rel='stylesheet' type='text/css'>
11.         <link rel="stylesheet" href="demo/css/demostyles.css">
12.         <link rel="stylesheet" href="css/simple-slideshow-styles.css">
13.     </head>
```

```

13.     <body>
14.         <!--[if lt IE 8]>
15.             <p class="browsehapp">You are using an <strong>outdated</strong> browser. Please <a
href="http://browsehapp.com/">upgrade your browser</a> to improve your experience.</p>
16.         <![endif]-->
17.         <header>
18.             <h1>A Better Simple Slideshow</h1>
19.             <p><span class="desc">A simple DIY responsive JavaScript slideshow.</span> [<a
href="https://github.com/leemark/better-simple-slideshow">GitHub<span> repo</span></a>]</p>
20.         </header>
21.         <div class="bss-slides num1" tabindex="1" autofocus="autofocus">
22.             <figure>
23.                 <figcaption>"Medium" by <a
href="https://www.flickr.com/photos/thomashawk/14586158819/">Thomas Hawk</a>.</figcaption>
24.             </figure>
25.             <figure>
26.                 <figcaption>"Colorado" by <a
href="https://www.flickr.com/photos/stuckincustoms/88370744">Trey Ratcliff</a>.</figcaption>
27.             </figure>
28.             <figure>
29.                 <figcaption>"Early Morning at
the Monte Vista Wildlife Refuge, Colorado" by <a
href="https://www.flickr.com/photos/davesoldano/8572429635">Dave Soldano</a>.</figcaption>
30.             </figure>
31.             <figure>
32.                 <figcaption>"Sunrise in Eastern
Colorado" by <a href="https://www.flickr.com/photos/35528040@N04/6673031153">Pam
Morris</a>.</figcaption>
33.             </figure>
34.             <figure>
35.                 <figcaption>"colorado
colors" by <a href="https://www.flickr.com/photos/cptspock/2857543585">Jasen
Miller</a>.</figcaption>
36.             </figure>
37.         </div> <!-- // bss-slides -->
38. <div class="content">
39. <h2>What is it?</h2>
40.
41. <p>It's a fairly basic slideshow, written in javascript. This is a dual-purpose project, it's
meant to be something you can drop right into your page and use if you so choose, but it's also
meant as an example/tutorial script showing how to build a simple DIY slideshow from scratch on
your own. <a href="http://themarklee.com/2014/10/05/better-simple-slideshow/">Here is a
tutorial/walkthrough</a>.</p>
42.
43. <h2>Features</h2>
44. <ul>
45.     <li>fully responsive</li>
46.     <li>option for auto-advancing slides, or manually advancing by user</li>
47.     <li>multiple slideshows per-page</li>
48.     <li>supports arrow-key navigation</li>
49.     <li>full-screen toggle using HTML5 fullscreen api</li>
50.     <li>swipe events supported on touch devices (requires <a
href="https://github.com/hammerjs/hammer.js">hammer.js</a>)</li>
51.     <li>written in vanilla JS--this means no jQuery dependency (much &hearts; for <a
href="https://github.com/jquery/jquery">jQuery</a> though!)</li>
52. </ul>
53.
54. <h2>Getting Started</h2>
55. <ol>
56. <li><p>HTML markup for the slideshow should look basically like this, with a container element
57. wrapping the whole thing (doesn't have to be a <span class="code">&lt;div&gt;</span>) and each

```



```

58. slide is a &lt;figure&gt;</span>.</p>
59.
60. <script src="https://gist.github.com/leemark/83571d9f8f0e3ad853a8.js"></script> </li>
61.
62. <li>Include the script: 

```





```

107. </div> <!-- // content -->
108. <footer>Example photos are property of their respective owners, all code is <a
109. href="https://github.com/leemark/better-simple-slideshow/blob/gh-pages/LICENSE">freely licensed
110. for your use</a>. <br>Made especially for you by <a href="http://themarklee.com">Mark Lee</a>
111. aka <a href="http://twitter.com/@therealmarklee">@therealmarklee</a> <br><span>&#9774; +
112. &hearts;</span></footer>
113. <script src="demo/js/hammer.min.js"></script><!-- for swipe support on touch interfaces -->
114. <script src="js/better-simple-slideshow.min.js"></script>
115. <script>
116. var opts = {
117.     auto : {
118.         speed : 3500,
119.         pauseOnHover : true
120.     },
121.     fullScreen : false,
122.     swipe : true
123. };
124. makeBSS('.num1', opts);
125. var opts2 = {
126.     auto : false,
127.     fullScreen : true,
128.     swipe : true
129. };
130. makeBSS('.num2', opts2);
131. </script>
132. </body>
133. </html>

```

As you can see, the full 'kitchen sink' of HTML web page capabilities are used in this example. The rendering is performed internally by IronPDF using the Chromium HTML engine and v8 javascript engine from Google. They do not need to be installed in your system, the entire package is automatically added to your project when you use IronPDF.

### 3.1. Add Headers and Footers

As we have a beautiful PDF render working, we may now wish to add attractive headers and footers

```

1. Imports IronPdf
2.
3. Module Module1
4.
5.     Sub Main()
6.         Dim renderer = New HtmlToPdf()
7.         renderer.PrintOptions.CssMediaType = PdfPrintOptions.PdfCssMediaType.Print
8.         renderer.PrintOptions.PrintHtmlBackgrounds = False
9.         renderer.PrintOptions.PaperOrientation = PdfPrintOptions.PdfPaperOrientation.Landscape
10.        renderer.PrintOptions.RenderDelay = 150
11.
12.        renderer.PrintOptions.Header.CenterText = "VB.Net PDF Slideshow"
13.        renderer.PrintOptions.Header.DrawDividerLine = True
14.        renderer.PrintOptions.Header.FontSize = "13"
15.
16.        renderer.PrintOptions.Footer.RightText = "page {page} of {total-pages}"
17.        renderer.PrintOptions.Footer.FontFamily = "Arial"
18.        renderer.PrintOptions.Footer.FontSize = "9"
19.

```



```

20.         Dim document = renderer.RenderHTMLFileAsPdf("../..\slideshow\index.html")
21.         document.SaveAs("Html5WithHeader.pdf")
22.         System.Diagnostics.Process.Start("Html5WithHeader.pdf")
23.     End Sub
24.
25. End Module

```

There is support for logical headers and footers as shown. You may also add HTML based headers and footers as described in the [VB.Net PDF developer object reference online](#)

You can download and explore [the source code for this "vb.net html to pdf"](#) project as a VB.Net Visual Studio project

## 4. Create PDF w/ Dynamic Content : 2 Methods

Historically, PDF 'templating' has been an overwhelming task for Software Engineers. Stamping content into PDF templates rarely works. This is because each case or report will contain content of varying types and length. Fortunately, HTML is exceptionally good at handling Dynamic Data.

For this we have 2 ways forward:

1. String Templating of HTML then conversion to PDF using .NET
2. Rendering out content as an ASP.NET Web Page and then rendering the page as a PDF

### 4.1. Method 1 - ASP.NET - ASPX to PDF using VB.Net Web Forms

Fortunately this solution is surprisingly simple. Any flavor of .Net Web Form (including Razor) can be rendered into a PDF document using this VB.Net code in the Page\_Load subroutine in the VB.Net code behind.

The PDF document may be set with a content-disposition to display in-browser, or to act as a file download.

```

1.     Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
2.         Dim PdfOptions As PdfPrintOptions = New IronPdf.PdfPrintOptions
3.         PdfOptions.
4.         IronPdf.AspxToPdf.RenderThisPageAsPDF(AspxToPdf.FileBehaviour.Attachment, "MyPdf.pdf",
5. PdfOptions)
6.     End Sub

```

### 4.2. Method 2 - HTML to PDF with String Templating

To create dynamic PDF documents that include instance specific data, we simply create a HTML string to match the data we wish to render as a PDF.

This is probably the largest advantage of the HTML-to-PDF solution in VB.Net - the ability to easily and intuitively create dynamic PDF documents and reports by creating HTML 'on the fly.'

The simplest version of this is the **String.Format** method from VB.Net

```
1. Imports IronPdf
2.
3. Module Module1
4.
5.     Sub Main()
6.         Dim renderer = New HtmlToPdf()
7.
8.         Dim Html = "Hello {0}"
9.         String.Format(Html, "World")
10.
11.        Dim document = renderer.RenderHtmlAsPdf("Html")
12.        document.SaveAs("HtmlTemplate.pdf")
13.        System.Diagnostics.Process.Start("HtmlTemplate.pdf")
14.    End Sub
15.
16. End Module
```

As PDFs get more complicated, the String will get more complicated. We might consider using a String Builder, or even a templating framework such as HandleBars.Net or Razor <https://github.com/rexm/Handlebars.Net>

---

## 5. Edit PDF Files with VB.Net

IronPDF for VB.Net also allows PDF documents to be edited, encrypted, watermarked or even turned back into plain text:

### 5.1. Merging Multiple PDF Files into One Document in VB

```
1. Dim Renderer As var = New IronPdf.HtmlToPdf
2. Dim PDFs As var = New List(Of PdfDocument)
3. PDFs.Add(PdfDocument.FromFile("A.pdf"))
4. PDFs.Add(PdfDocument.FromFile("B.pdf"))
5. PDFs.Add(PdfDocument.FromFile("C.pdf"))
6. Dim PDF As PdfDocument = PdfDocument.Merge(PDFs)
7. PDF.SaveAs("merged.pdf")
```

### 5.2. Add a Cover Page to the PDF

```
1. PDF.PrependPdf(Renderer.RenderHtmlAsPdf("<h1>Cover Page</h1><hr>"))
```

### 5.3. Remove the last page from the PDF

```
1. PDF.RemovePage((PDF.PageCount - 1))
```

## 5.4. Encrypt a PDF using 128 Bit Encryption

```
1. // Save with a strong encryption password.
2. PDF.Password = "my.secure.password";
3. PDF.SaveAs("secured.pdf")
```

## 5.5. Stamp Additional HTML Content Onto a Page in VB

```
1. Imports IronPdf
2.
3. Module Module1
4.
5.     Sub Main()
6.         Dim Renderer As IronPdf.HtmlToPdf = New IronPdf.HtmlToPdf
7.         Dim pdf = Renderer.RenderUrlAsPdf("https://www.nuget.org/packages/IronPdf")
8.
9.         Dim stamp = New HtmlStamp()
10.        stamp.Html = "<h2>Completed</h2>"
11.        stamp.Opacity = 50
12.        stamp.Rotation = -45
13.        stamp.Top = 10
14.        pdf.StampHTML(stamp)
15.
16.        pdf.SaveAs("C:\Path\To\Stamped.pdf")
17.    End Sub
18.
19. End Module
```

## 5.6. Add Page Break to PDF Using HTML

The easiest way to do this is with HTML and CSS

```
1. <div style='page-break-after: always;'>&nbsp;</div>
```

---

## 6. More .NET PDF Tutorials

You may also be interested in:

- [The full VB.Net and C# MSDN style object reference](#)
- [A tutorial about converting ASPX to PDF for Vb.Net and C#](#)
- [An in depth tutorial about rendering HTML to PDF for Vb.Net and C#](#)

---

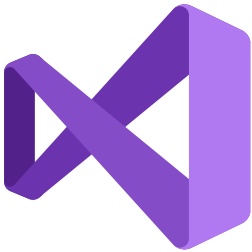
## Conclusion

In this tutorial we discovered 6 ways to achieve VB.Net to PDF results using VB.NET as our programming language of choice.

- HTML string to PDF
- Creating a PDF in VB.Net using an HTML string to define its content
- Rendering existing URLs as PDF files
- Generating PDF from HTML files
- HTML templating in VB.Net and conversion to dynamic PDFs
- Converting ASP.Net pages with live data, such as ASPX to PDF files

For each we used the popular IronPDF VB .NET library to allow us to turn HTML directly into PDF documents within .NET projects

## Tutorial Quick Access



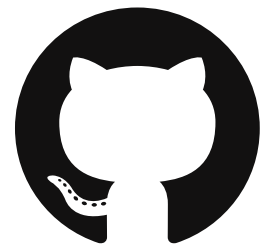
### Download this Tutorial as C# Source Code

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

[Download](#)

### Explore this Tutorial on GitHub

You may also be interested in our extensive library of VB.Net PDF generation and manipulation examples on GitHub. Exploring source code is the fastest way to learn, and Github is the definitive way to do so online. I hope these examples help you get to grips with PDF related functionality in your VB projects.



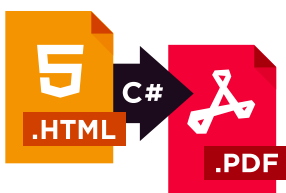
[Creating PDFs in ASP.Net with VB.Net and C# Source](#) >

[A Simple Hello World Project to Render HTML to PDF in VB.Net using Iron PDF](#) >

[Exploring Html To PDF in-depth with VB.Net](#) >

### 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" provides quick access to common functions and examples for generating and editing PDFs in C# and VB.Net - and will save time 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](#)