Bruno Lowagie

iText 7

Converting HTML to PDF with pdfHTML

Create PDF from HTML with pdfHTML

Generate PDF documents using HTML and CSS as template format

iText Software

This book is for sale at http://leanpub.com/itext7_pdfHTML

This version was published on 2017-08-30

Tweet This Book!

Please help iText Software by spreading the word about this book on Twitter!

The suggested hashtag for this book is #iText7_pdfHTML.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#iText7_pdfHTML

Contents

Introduction

In this tutorial, we'll learn how to convert HTML to PDF using pdfHTML, an add-on to iText 7. If you're new to iText, please jump to chapter 1 immediately. If you've been working with iText in the past, you might remember the old HTML to PDF functionality. If that's the case, you've either been using the obsolete `HTMLWorker` class (iText 2), or the old XML Worker add-on (iText 5).

The `HTMLWorker` class was deprecated many years ago. The goal of `HTMLWorker` was to convert small, simple HTML snippets to iText objects. It was never meant to convert complete HTML pages to PDF, yet that was how many developers tried to use it. This caused plenty of frustration because `HTMLWorker` didn't support every HTML tag, didn't parse CSS files, and so on. To avoid this frustration, `HTMLWorker` was removed from recent versions of iText.

In 2011, iText Group released XML Worker as a generic XML to PDF tool, built on top of iText 5. A default implementation converted XHTML (data) and CSS (styles) to PDF, mapping HTML tags such as `<p>`, `<img>`, and `<li>` to iText 5 objects such as `Paragraph`, `Image`, and `ListItem`. We don't know of any implementations that used XML Worker for any other XML formats, but many developers used XML Worker in combination with jsoup as an HTML2PDF converter.

XML Worker wasn't a URL2PDF tool though. XML Worker expected predictable HTML created for the sole purpose of converting that HTML to PDF. A common use case was the creation of invoices. Rather than programming the design of an invoice in Java or C#, developers chose to create a simple HTML template defining the structure of the document, and some CSS defining the styles. They then populated the HTML with data, and used XML Worker to create the invoices as PDF documents, throwing away the original HTML. We'll take a closer look at this use case in chapter 4, converting XML to HTML in memory using XSLT, then converting that HTML to PDF using the pdfHTML add-on.

When iText 5 was originally created, it was designed as a tool to produce PDF as fast as possible, flushing pages to the `OutputStream` as soon as they were finished. Several design choices that made perfect sense when iText was first released in the year 2000, were still present in iText 5 sixteen years later. Unfortunately, some of these choices made it very difficult –if not impossible– to extend the functionality of XML Worker to the level of quality many developers expected. If we really wanted to create a great HTML to PDF converter, we would have to rewrite iText from scratch. Which we did.

In 2016, we released iText 7, a brand new version of iText that was no longer compatible with previous versions, but that was created with pdfHTML in mind. A lot of work was spent on the new `Renderer` framework. When a document is created with iText 7, a tree of renderers and their child-renderers is built. The layout is created by traversing that tree, an approach that is much better suited when dealing with HTML to PDF conversion. The iText objects were completely redesigned to better match HTML tags and to allow setting styles "the CSS way."

For instance: in iText 5, you had a `PdfPTable` and a `PdfPCell` object to create a table and its cells. If you wanted every cell to contain text in a font different from the default font, you needed to set that font for the content of every separate cell. In iText 7, you have a `Table` and `Cell` object, and when you set a different font for the complete table, this font is inherited as the default font for every cell. That was a major step forward in terms of architectural design, especially if the goal is to convert HTML to PDF.

But let's not dwell on the past, let's see what pdfHTML can do for us. In the first chapter, we'll take a look at different variations of the `convertToPdf()` method, and we'll discover how the converter is configured.

Chapter 1: Hello HTML to PDF

In this chapter, we'll convert a simple HTML file to a PDF document in many different ways. The content of the HTML file will consist of a "Test" header, a "Hello World" paragraph, and an image representing the iText logo.

Structure of the examples

All the examples throughout this book will have a similar structure.

INPUT:

For the input, we'll provide HTML syntax. In this tutorial, we'll use an HTML `String`, a path to an HTML file, or –in chapter 4– a path to an XML file along with the path to an XSLT file to convert the XML to HTML.

In the first example, C01E01_HelloWorld, the HTML is provided as a `String`:

```
public static final String HTML = "<h1>Test</h1><p>Hello World</p>";
```

In other examples, such as C01E03_HelloWorld, we'll use two constants:

- a `BASEURI` constant for the path to the parent folder where to find the source HTML and resources such as images and CSS, and
- a `SRC` constant with the path to that source HTML file.

For instance:

```
public static final String BASEURI = "src/main/resources/html/";
public static final String SRC = String.format("%shello.html", BASEURI);
```

OUTPUT:

We'll use a similar structure for the output:

- a `TARGET` constant for the path to the folder to which we'll write the resulting PDF, and
- a `DEST` constant with the path to that PDF.

For instance:

```
public static final String TARGET = "target/results/ch01/";
public static final String DEST = String.format("%stest-03.pdf", TARGET);
```

MAIN METHOD:

The `main()` method of all the examples in this book won't differ much from the `main()` method of our first example:

```
1 public static void main(String[] args) throws IOException {
2     LicenseKey.loadLicenseFile(
3         System.getenv("ITEXT7_LICENSEKEY") + "/itextkey-html2pdf_typography.xml");
4     File file = new File(TARGET);
5     file.mkdirs();
6     new C01E01_HelloWorld().createPdf(HTML, DEST);
7 }
```

First we load the iText license file (line 2-3). This is an XML file containing a license key for using iText. You might not need this license key if you are using iText and pdfHTML in the context of an AGPL project. However, you will need the pdfCalligraph add-on for the internationalization examples in chapter 6, and the pdfCalligraph add-on isn't available under the AGPL; it's a closed source add-on only.

The license key we are using in the examples of this book is similar to the key you will get if you purchase a commercial license to use iText 7, pdfHTML, and pdfCalligraph in a closed source context.

In lines 4 and 5, we create the target directory in case it doesn't exist yet. In line 6, we call the `createPdf()` method. We can implement this methods in many different ways.

Converting HTML to PDF

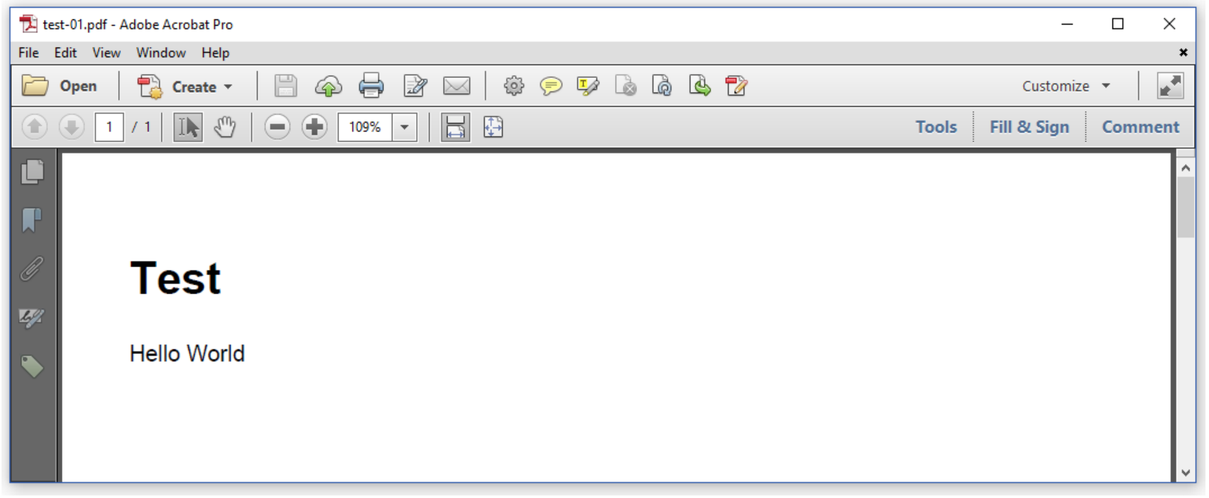
The implementation of the `createPdf()` method of the C01E01_HelloWorld example is very simple. Its body consists of a single line:

```
public void createPdf(String html, String dest) throws IOException {
    HtmlConverter.convertToPdf(html, new FileOutputStream(dest));
}
```

The `HtmlConverter` object has a selection of different static `convertToPdf()` methods that take different parameters depending on the use case. In the first example, the first parameter `html` is a `String` with the following value:

```
public static final String HTML = "<h1>Test</h1><p>Hello World</p>";
```

This HTML snippet is converted to the PDF document that is shown in figure 1.1.



Test

Hello World

Figure 1.1: converting an HTML snippet to PDF

Let's introduce an image, and use the following `String`:

```
public static final String HTML =
    "<h1>Test</h1><p>Hello World</p><img src=\"img/logo.png\">";
```

This HTML snippet contains a relative link to the image file `logo.png` in a subdirectory named `img`. It's impossible for iText to guess where to look for this subdirectory, hence we'll configure the base URI for the conversion process.

This is done using the `ConverterProperties` object, as shown in the `createPdf()` method of the C01E02_-HelloWorld example.

```
public void createPdf(String baseUri, String html, String dest) throws IOException {
    ConverterProperties properties = new ConverterProperties();
    properties.setBaseUri(baseUri);
    HtmlConverter.convertToPdf(html, new FileOutputStream(dest), properties);
}
```

We create a `ConverterProperties` object, and we set the base URI to the parent directory of the `img` directory where iText can find the `logo.png` file.

Figure 1.2 shows the result.

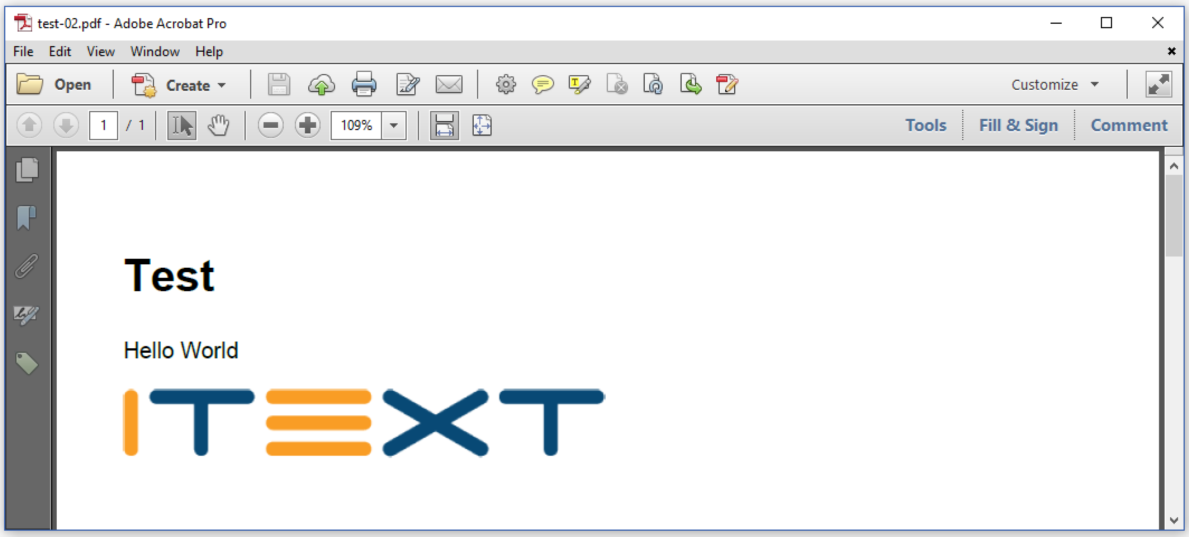
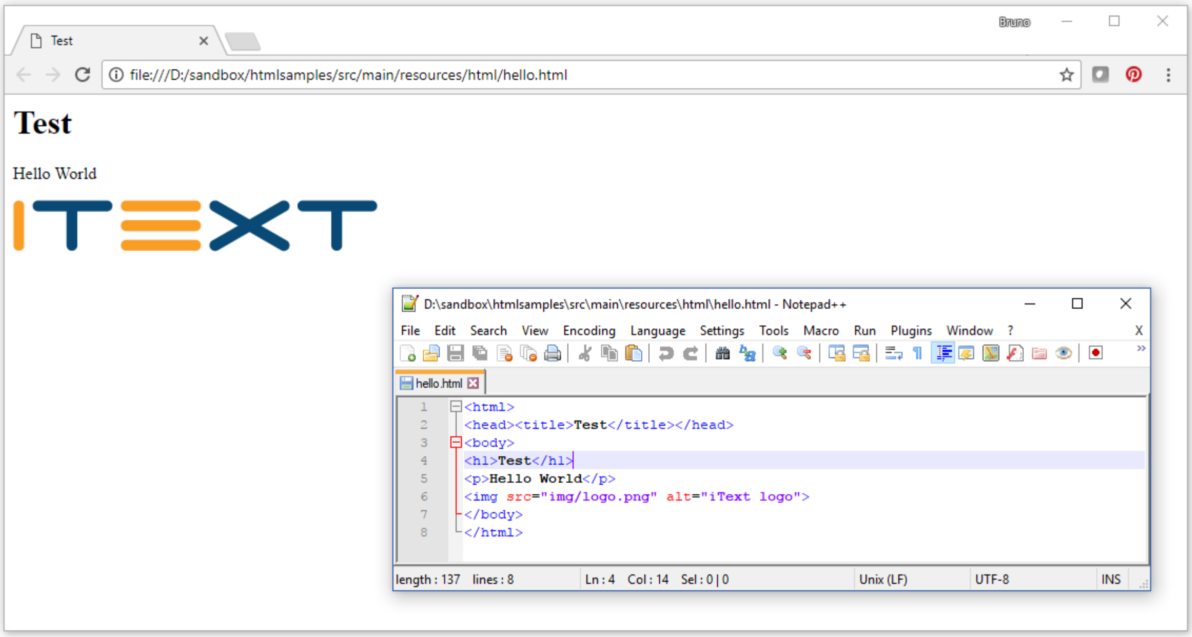


Figure 1.2: converting an HTML snippet containing a reference to an image

In most of the examples that follow, we won't use HTML stored in a `String`. Instead, we are going to convert an HTML file on disk into a file on disk.

For the rest of the examples in this chapter, we'll use the file named `hello.html` shown in figure 1.3.


Test
Bruno
file:///D:/sandbox/htmlsamples/src/main/resources/html/hello.html
Test
Hello World
D:\sandbox\htmlsamples\src\main\resources\html\hello.html - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
hello.html

```
<html>
<head><title>Test</title></head>
<body>
<h1>Test</h1>
<p>Hello World</p>
<img src="img/logo.png" alt="iText logo">
</body>
</html>
```

```
length : 137   lines : 8   Ln : 4   Col : 14   Sel : 0 | 0   Unix (LF)   UTF-8   INS
```

Figure 1.3: hello.html shown in a browser as well as in a text editor

There are different ways to convert this file to a PDF document.

In the C01E03_HelloWorld.java example, we use `File` objects:

```java
public void createPdf(String baseUri, String src, String dest) throws IOException {
    HtmlConverter.convertToPdf(new File(src), new File(dest));
}
```

The first parameter of the `convertToPdf()` method refers to the source HTML file, the second parameter to the destination PDF file. In this case, we don't need to set any converter properties. If `file` is the `File` object of the HTML file, iText uses `file.getParent()` to get the parent directory, and uses this parent directory as the base URI.

This doesn't work for the C01E04_HelloWorld example where we use `FileInputStream` and `FileOutputStream` objects instead of `File` objects:

```
public void createPdf(String baseUri, String src, String dest) throws IOException {
    ConverterProperties properties = new ConverterProperties();
    properties.setBaseUri(baseUri);
    HtmlConverter.convertToPdf(
        new FileInputStream(src), new FileOutputStream(dest), properties);
}
```

You can't retrieve a parent from an `InputStream`, hence we need to pass a base URI to the converter using a `ConverterProperties` instance. The resulting PDFs of this third and fourth example look identical to the resulting PDF of the second example shown in figure 1.2. So does the resulting PDF of the fifth example, C01E05_HelloWorld:

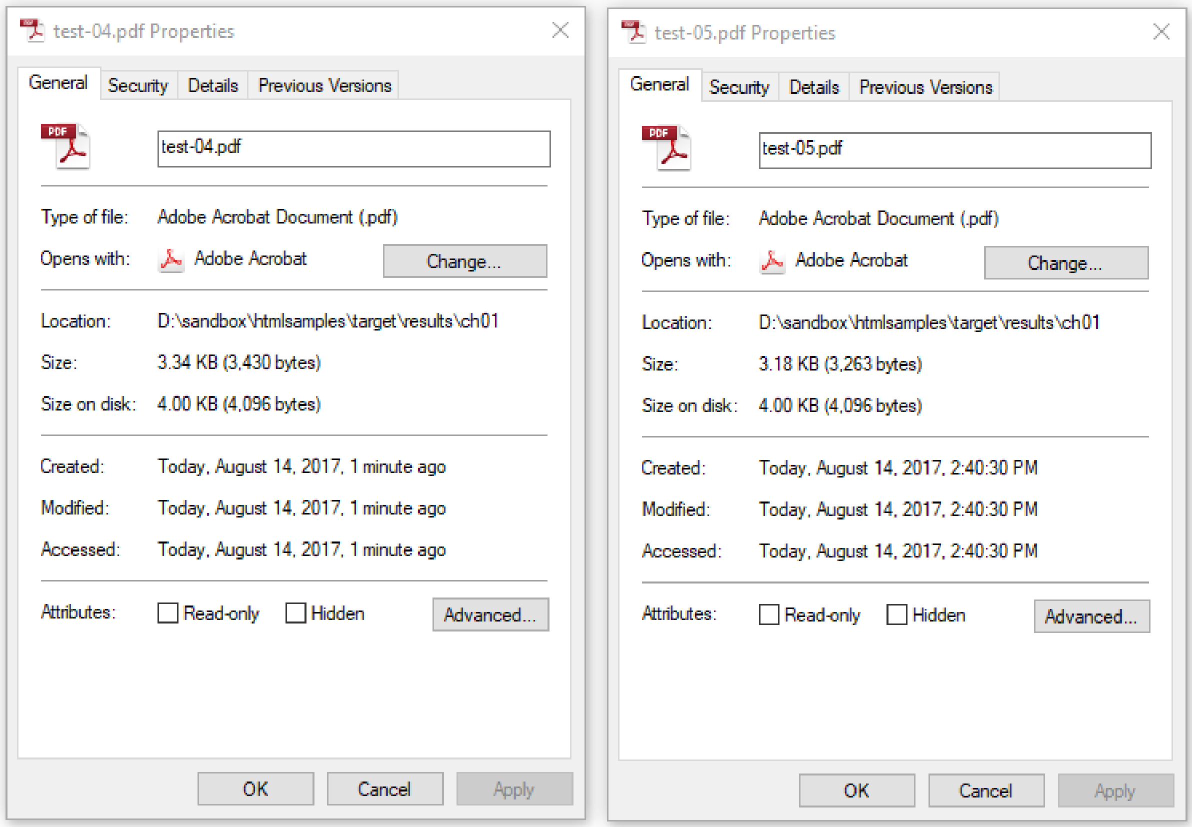
```
public void createPdf(String baseUri, String src, String dest) throws IOException {
    ConverterProperties properties = new ConverterProperties();
    properties.setBaseUri(baseUri);
    PdfWriter writer = new PdfWriter(dest,
        new WriterProperties().setFullCompressionMode(true));
    HtmlConverter.convertToPdf(new FileInputStream(src), writer, properties);
}
```

In this case, we use a `PdfWriter` instance instead of a `FileOutputStream`. Using a `PdfWriter` can be useful if you want to set certain writer properties.

For more information on writer properties, please read Chapter 7 of the iText 7: Building Blocks tutorial, entitled "Handling events; setting viewer preferences and printer properties."

In this example, we create the PDF in *full compression mode*. To the human eye, the resulting PDF looks identical, but when you compare the file size of the PDF generated in example 4 with the file size of the PDF generated in this example, you see that full compression won us a handful of bytes.

Figure 1.4 shows 3,430 bytes when using compression as was done in PDF 1.0 to PDF 1.4; whereas the file only counts 3,263 bytes when using compression as introduced in PDF 1.5. That difference might seem small, but the more objects your PDF has, the more sense it makes to use full compression.



test-04.pdf Properties

General | Security | Details | Previous Versions

test-04.pdf

Type of file: Adobe Acrobat Document (.pdf)
Opens with: Adobe Acrobat
Location: D:\sandbox\htmlsamples\target\results\ch01
Size: 3.34 KB (3,430 bytes)
Size on disk: 4.00 KB (4,096 bytes)
Created: Today, August 14, 2017, 1 minute ago
Modified: Today, August 14, 2017, 1 minute ago
Accessed: Today, August 14, 2017, 1 minute ago
Attributes: ☐ Read-only ☐ Hidden

test-05.pdf Properties

General | Security | Details | Previous Versions

test-05.pdf

Type of file: Adobe Acrobat Document (.pdf)
Opens with: Adobe Acrobat
Location: D:\sandbox\htmlsamples\target\results\ch01
Size: 3.18 KB (3,263 bytes)
Size on disk: 4.00 KB (4,096 bytes)
Created: Today, August 14, 2017, 2:40:30 PM
Modified: Today, August 14, 2017, 2:40:30 PM
Accessed: Today, August 14, 2017, 2:40:30 PM
Attributes: ☐ Read-only ☐ Hidden

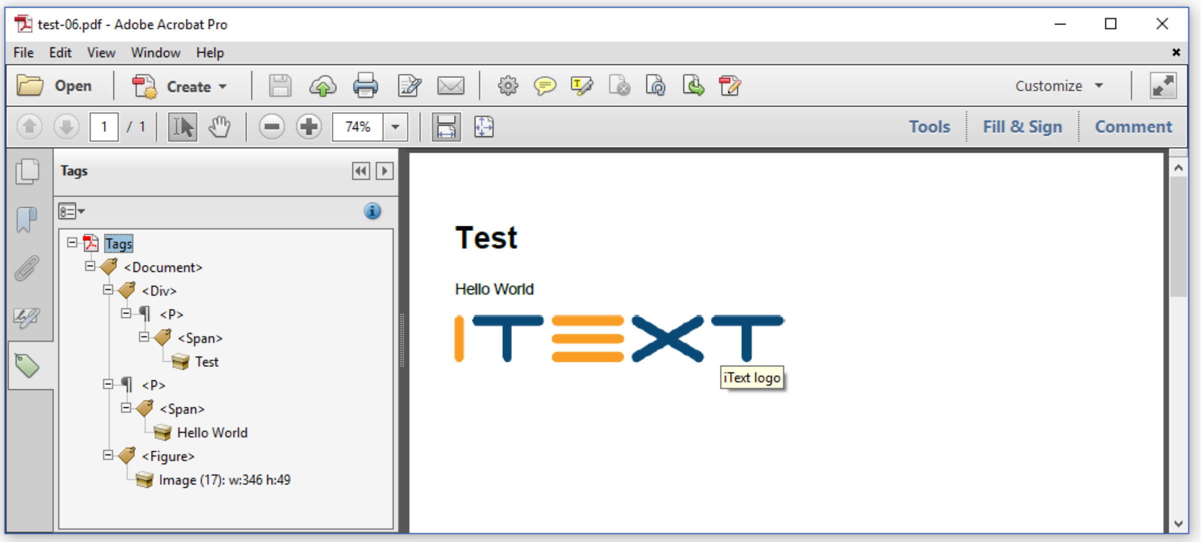
Figure 1.4: comparing file sizes

In the C01E06_HelloWorld example, we've replaced the `PdfWriter` parameter with a `PdfDocument` parameter.

```
public void createPdf(String baseUri, String src, String dest) throws IOException {
    ConverterProperties properties = new ConverterProperties();
    properties.setBaseUri(baseUri);
    PdfWriter writer = new PdfWriter(dest);
    PdfDocument pdf = new PdfDocument(writer);
    pdf.setTagged();
    HtmlConverter.convertToPdf(new FileInputStream(src), pdf, properties);
}
```

Using a `PdfDocument` instance makes sense if you want to configure a feature at the `PdfDocument` level. In this case, we introduce the line `pdf.setTagged()`, which instructs iText to create a Tagged PDF.

Figure 1.5 shows the resulting PDF with the Tags panel opened.



Tags
<Document>
<Div>
<P>
<Span>
Test
<P>
<Span>
Hello World
<Figure>
Image (17): w:346 h:49

Figure 1.5: creating Tagged PDF

Looking at the Tags panel, you can see the structure of the content. When hovering over the image, you see the value of the `alt` attribute of the `<img>`-tag as a tooltip.

For more info on Tagged PDF, please read Chapter 7 of the iText 7: Jump-Start Tutorial, entitled “Creating PDF/UA and PDF/A documents.”

We’ll dive deeper into Tagged PDF and making PDFs “accessible” in chapter 3.

Converting HTML to iText objects

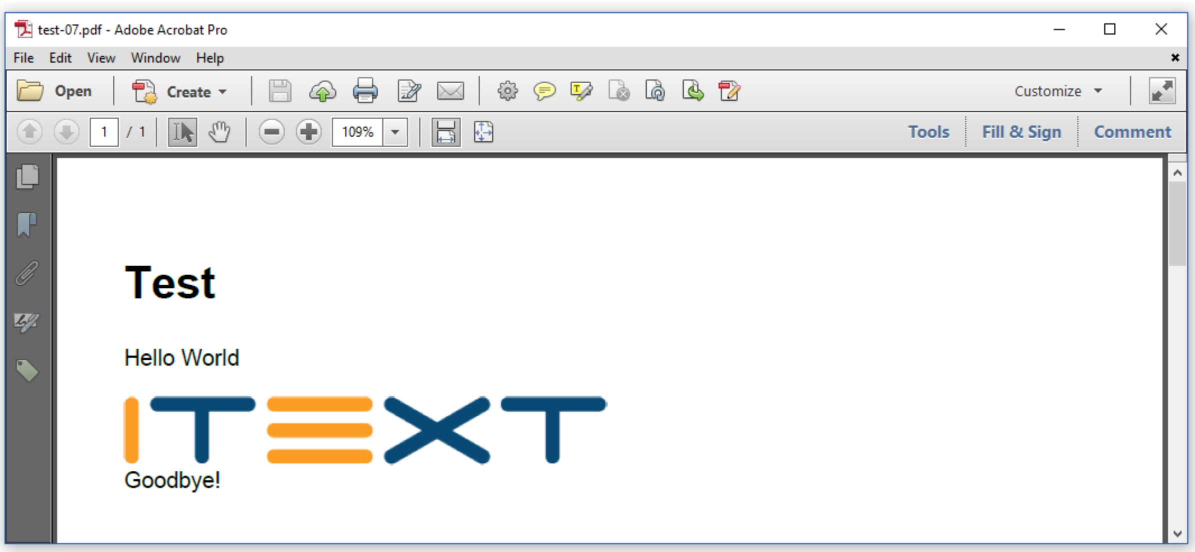
The `convertToPdf()` methods create a complete PDF file. Any `File`, `OutputStream`, `PdfWriter`, or `PdfDocument` that is passed to the `convertToPdf()` method is closed once the input is parsed and converted to PDF. This might not always be what you want.

In some cases, you want to add some extra information to the `Document`, or maybe you don’t want to convert the HTML to a PDF file, but to a series of iText objects you can use for a different purpose. That’s what the `convertToDocument()` and `convertToElements()` methods are about.

In the C01E07_HelloWorld example, we convert our Hello World HTML to a `Document` because we want to add some extra content after we’ve done parsing the HTML:

```
public void createPdf(String baseUri, String src, String dest) throws IOException {
    ConverterProperties properties = new ConverterProperties();
    properties.setBaseUri(baseUri);
    PdfWriter writer = new PdfWriter(dest);
    PdfDocument pdf = new PdfDocument(writer);
    Document document =
        HtmlConverter.convertToDocument(new FileInputStream(src), pdf, properties);
    document.add(new Paragraph("Goodbye!"));
    document.close();
}
```

The `convertToDocument()` method returns an iText `Document` instance. We use this `Document` instance to add some extra content (`"Goodbye!"`) after the HTML has been parsed.



Test

Hello World

Goodbye!

Figure 1.6: using the convertToDocument() method

The upper part of the content in figure 1.6 was added by parsing HTML to PDF; the lower part –the "Goodbye!" at the end– was added using a `document.add()` instruction.

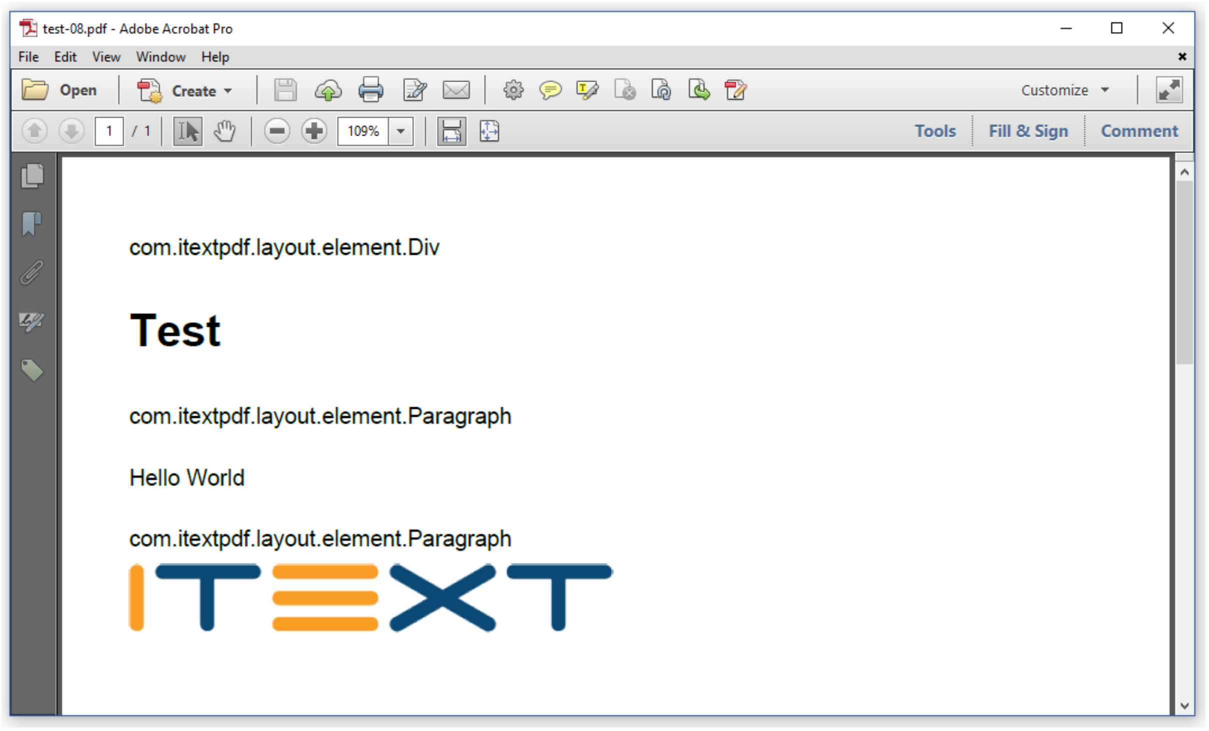
In the C01E08_HelloWorld example, we use the `convertToElements()` method. This method creates a `List` of `IElement` objects. The `IElement` interface is implemented by all the iText building blocks.

For more info about iText's building blocks, please read the iText 7: Building Blocks tutorial.

This last example of chapter 1 adds every top-level object of the `List<IElement>` collection to a `Document`, preceded by a `Paragraph` that shows the name of that object:

```
public void createPdf(String baseUri, String src, String dest) throws IOException {
    ConverterProperties properties = new ConverterProperties();
    properties.setBaseUri(baseUri);
    List<IElement> elements =
        HtmlConverter.convertToElements(new FileInputStream(src), properties);
    PdfDocument pdf = new PdfDocument(new PdfWriter(dest));
    Document document = new Document(pdf);
    for (IElement element : elements) {
        document.add(new Paragraph(element.getClass().getName()));
        document.add((IBlockElement)element);
    }
    document.close();
}
```

Looking at figure 1.7, we see that the list consisted of three elements: one `Div` and two `Paragraph` objects.



com.itextpdf.layout.element.Div

Test

com.itextpdf.layout.element.Paragraph

Hello World

com.itextpdf.layout.element.Paragraph

Figure 1.7: adding elements one at a time

The header is treated as a `Div`, whereas the logo image is wrapped inside a `Paragraph`. Don't worry about this; this is part of the inner workings of iText. It's the end result that matters.

Summary

In this chapter, we've taken one very simple HTML file, and we've converted that file to PDF using different implementations of the conversion methods `convertToPdf()`, `convertToDocument()`, and `convertToElements()`. When you consult the API documentation for the HtmlConverter class, you'll discover some more variations on those methods. In the next chapter, we'll pick one of those methods to convert different HTML files. Each of these HTML files will use CSS in a different way.

Chapter 2: Defining styles with CSS

In the previous chapter, we looked at different snippets of Java code.

In this chapter, we'll use the same snippet for every example:

```java
public void createPdf(String src, String dest) throws IOException {
    HtmlConverter.convertToPdf(new File(src), new File(dest));
}
```

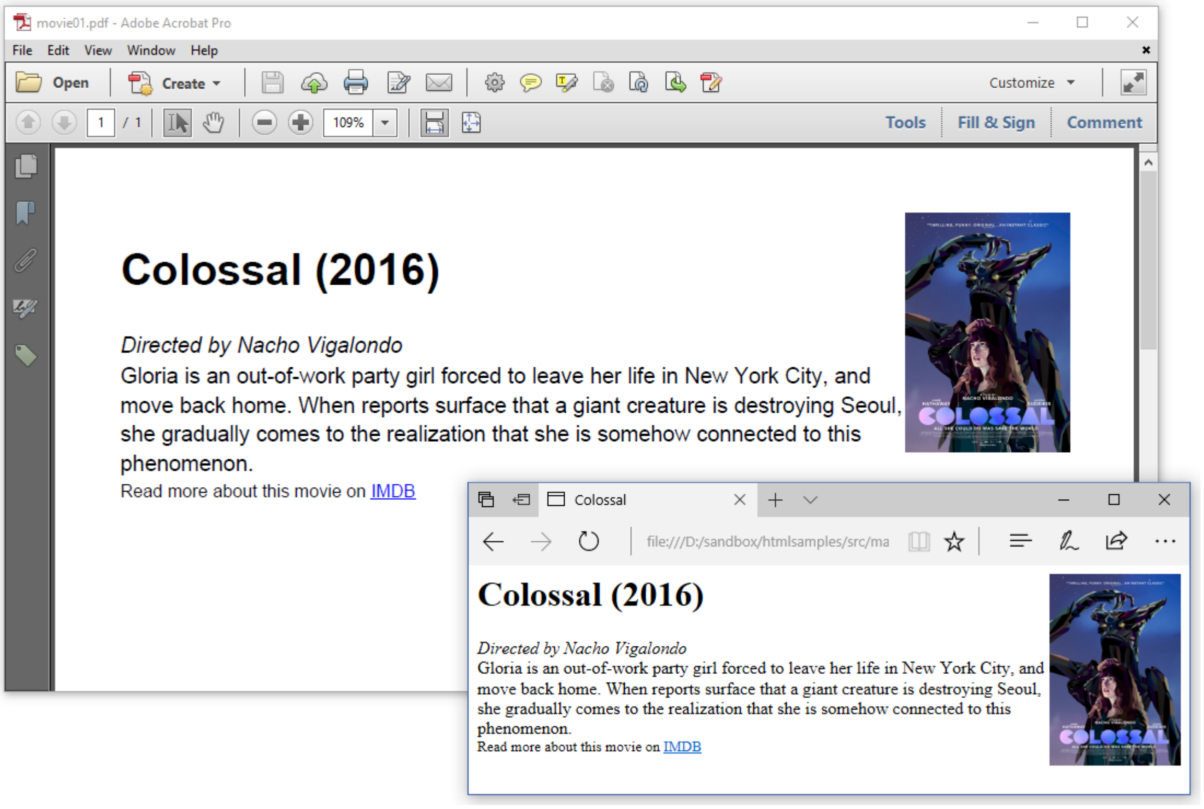
Instead of looking at different snippets of Java code, we'll look at different snippets of HTML and CSS.

Old-fashioned HTML

In example C02E01_NoCss, we define styles as *italic* and a different font size by using tags such as `<i>` and `<font>`; see 1_no_css.html:

```html
<html>
    <head>
        <title>Colossal</title>
        <meta name="description" content="Gloria is an out-of-work party girl ..." />
    </head>
    <body>
        <img src="img/colossal.jpg" width="120px" align="right" />
        <h1>Colossal (2016)</h1>
        <i>Directed by Nacho Vigalondo</i>
        <div>
        Gloria is an out-of-work party girl forced to leave her life in New York City,
        and move  back home. When reports surface that a giant creature is
        destroying Seoul, she gradually comes to the realization that she is
        somehow connected to this phenomenon.
        </div>
        <font size="-1">Read more about this movie on
        <a href="www.imdb.com/title/tt4680182">IMDB</a></font>
    </body>
</html>
```

The HTML page and the PDF that were created based on this HTML page are shown in figure 2.1.


movie01.pdf - Adobe Acrobat Pro
File Edit View Window Help
Open
Create
Customize
Tools
Fill & Sign
Comment
Colossal
file:///D:/sandbox/htmlsamples/src/ma

Colossal (2016)

Directed by Nacho Vigalondo
Gloria is an out-of-work party girl forced to leave her life in New York City, and move back home. When reports surface that a giant creature is destroying Seoul, she gradually comes to the realization that she is somehow connected to this phenomenon.
Read more about this movie on IMDB

Figure 2.1: HTML page without CSS

We call this old-fashioned –some would call it bad taste– because nowadays HTML is only used to define the content and its structure. Nowadays, all styles –such as widths, heights, the choice of font, font size, font color, font weight, and so on…– are defined using Cascading Style Sheets (CSS). This is a much more elegant approach, as it creates a clear separation between the content and its presentation.

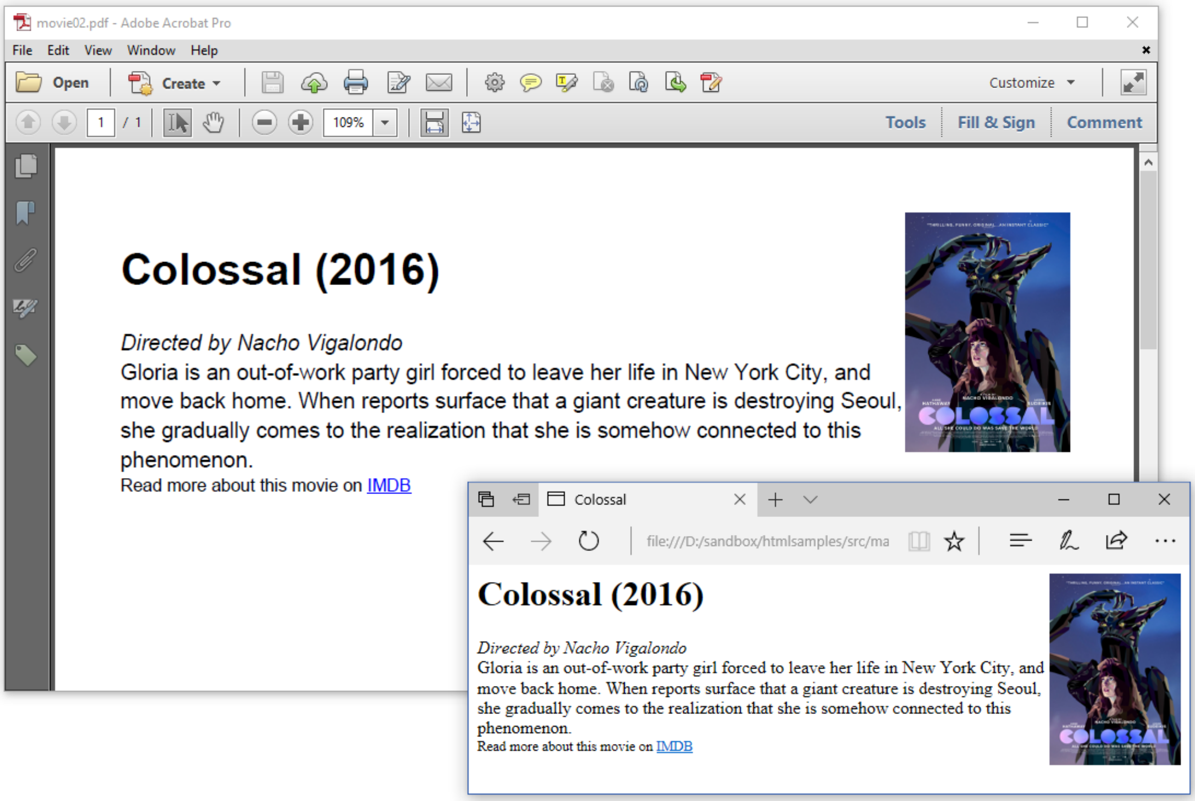
Let’s rewrite our HTML page, and let’s introduce some CSS.

Inline CSS

When opened in a browser, there is no visible difference between the HTML file from the previous example, and 2_inline_css.html:

```
<html>
    <head>
        <title>Colossal</title>
        <meta name="description" content="Gloria is an out-of-work party girl ..." />
    </head>
    <body>
        <img src="img/colossal.jpg" style="width: 120px;float: right" />
        <h1>Colossal (2016)</h1>
        <div style="font-style: italic">Directed by Nacho Vigalondo</div>
        <div>
        Gloria is an out-of-work party girl forced to leave her life in New York City,
        and move  back home. When reports surface that a giant creature is
        destroying Seoul, she gradually comes to the realization that she is
        somehow connected to this phenomenon.
        </div>
        <div style="font-size: 0.8em">Read more about this movie on
        <a href="www.imdb.com/title/tt4680182">IMDB</a></div>
    </body>
</html>
```

The width of the image and its position are now defined in a `style` attribute. So is the font style used for the director, as well as the font size for the IMDB link. As you can tell from figure 2.2, there is no difference between the output of this example and the previous one.


movie02.pdf - Adobe Acrobat Pro
File Edit View Window Help
Open
Create
Customize
1 / 1
109%
Tools
Fill & Sign
Comment
Colossal
file:///D:/sandbox/htmlsamples/src/ma

Colossal (2016)

Directed by Nacho Vigalondo
Gloria is an out-of-work party girl forced to leave her life in New York City, and move back home. When reports surface that a giant creature is destroying Seoul, she gradually comes to the realization that she is somehow connected to this phenomenon.
Read more about this movie on IMDB

Colossal (2016)

Directed by Nacho Vigalondo
Gloria is an out-of-work party girl forced to leave her life in New York City, and move back home. When reports surface that a giant creature is destroying Seoul, she gradually comes to the realization that she is somehow connected to this phenomenon.
Read more about this movie on IMDB

Figure 2.2: HTML page with inline CSS

Instead of using the `style` attribute, we can also use the `id` or the `class` attribute. Both are supported by pdfHTML.

What's the difference between id and class?

- An `id` is unique: each element can have only one id; each page can have only one element with that id.
- A `class` is not unique: you can use the same class on multiple elements; you can use multiple classes on the same element.

In the next example, we'll define some classes, and we'll put them in the header section of the HTML page.

Internal CSS

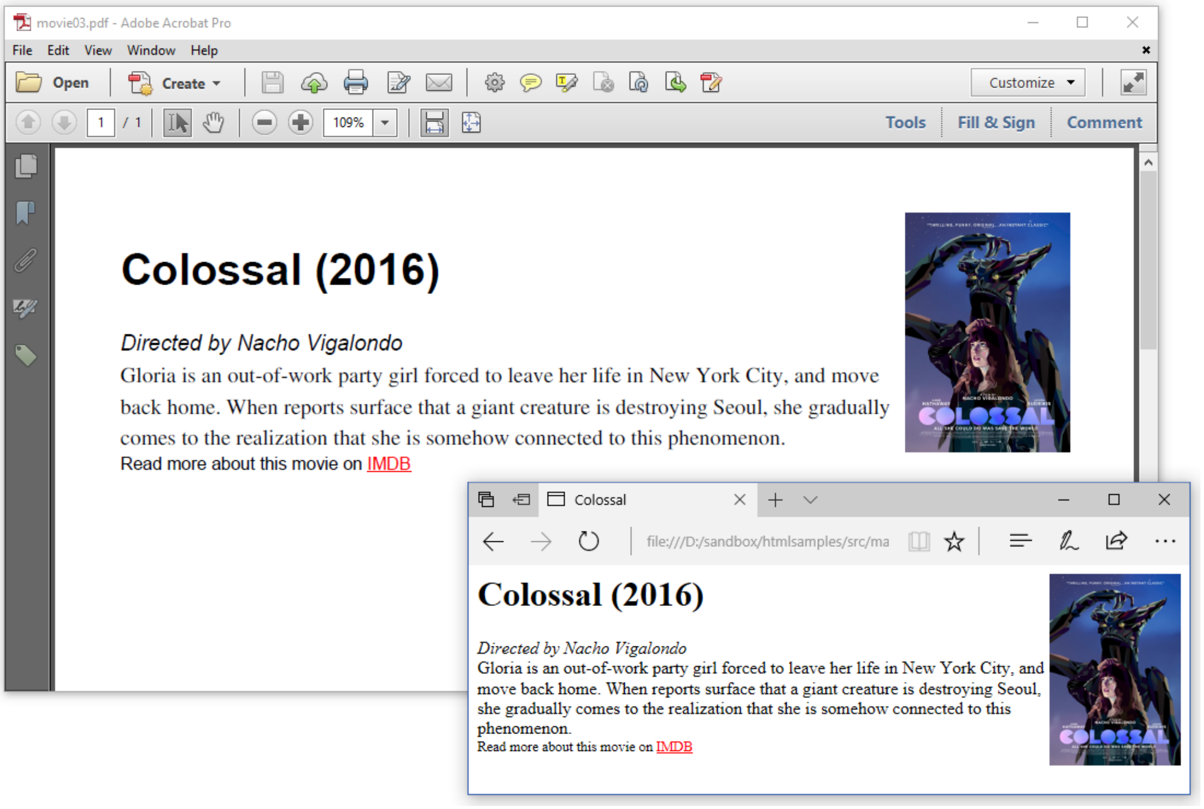
In the 3_header_css.html HTML file, we bundle all the styles in a `<style>` section in the header of the HTML file.

```html
<html>
    <head>
        <title>Colossal</title>
        <meta name="description" content="Gloria is an out-of-work party girl ..." />
        <style>
            .poster { width: 120px;float: right; }
            .director { font-style: italic; }
            .description { font-family: serif; }
            .imdb { font-size: 0.8em; }
            a { color: red; }
        </style>
    </head>
    <body>
        <img src="img/colossal.jpg" class="poster" />
        <h1>Colossal (2016)</h1>
        <div class="director">Directed by Nacho Vigalondo</div>
        <div class="description">
        Gloria is an out-of-work party girl forced to leave her life in New York City,
        and move  back home. When reports surface that a giant creature is
        destroying Seoul, she gradually comes to the realization that she is
        somehow connected to this phenomenon.
        </div>
        <div class="imdb">Read more about this movie on
        <a href="www.imdb.com/title/tt4680182">IMDB</a></div>
    </body>
</html>
```

We made a couple of changes when compared to the previous two examples. We changed the color of the links to red (instead of using the default color, which is blue), and we picked a `serif` font for the description (instead of using the default font).

When you don't define a font in an HTML file, most browsers will show the document in a serif font. Historically, the default font used by iText has always been Helvetica, which is a sans-serif font. This explains the difference between the original HTML files and the resulting PDFs in the examples so far.

You could define the `font-family` as `serif` at the level of the `body` element to have a better match between the HTML and the PDF. We chose not to do this in this simple example; as a result, we can clearly see the difference between the serif and the sans-serif fonts in figure 2.3:


movie03.pdf - Adobe Acrobat Pro
File Edit View Window Help
Open
Create
Customize
Tools
Fill & Sign
Comment
Colossal
file:///D:/sandbox/htmlsamples/src/ma

Colossal (2016)

Directed by Nacho Vigalondo

Gloria is an out-of-work party girl forced to leave her life in New York City, and move back home. When reports surface that a giant creature is destroying Seoul, she gradually comes to the realization that she is somehow connected to this phenomenon.

Read more about this movie on IMDB

Figure 2.3: HTML page with CSS in the header section

The title, the line with the name of the director, and the IMDB paragraph are still in Helvetica, but the description is rendered in a Serif font.

Putting CSS in the header is fine, but as soon as more styles are involved, it might be better to put the CSS in a separate file, external to the HTML file.

External CSS

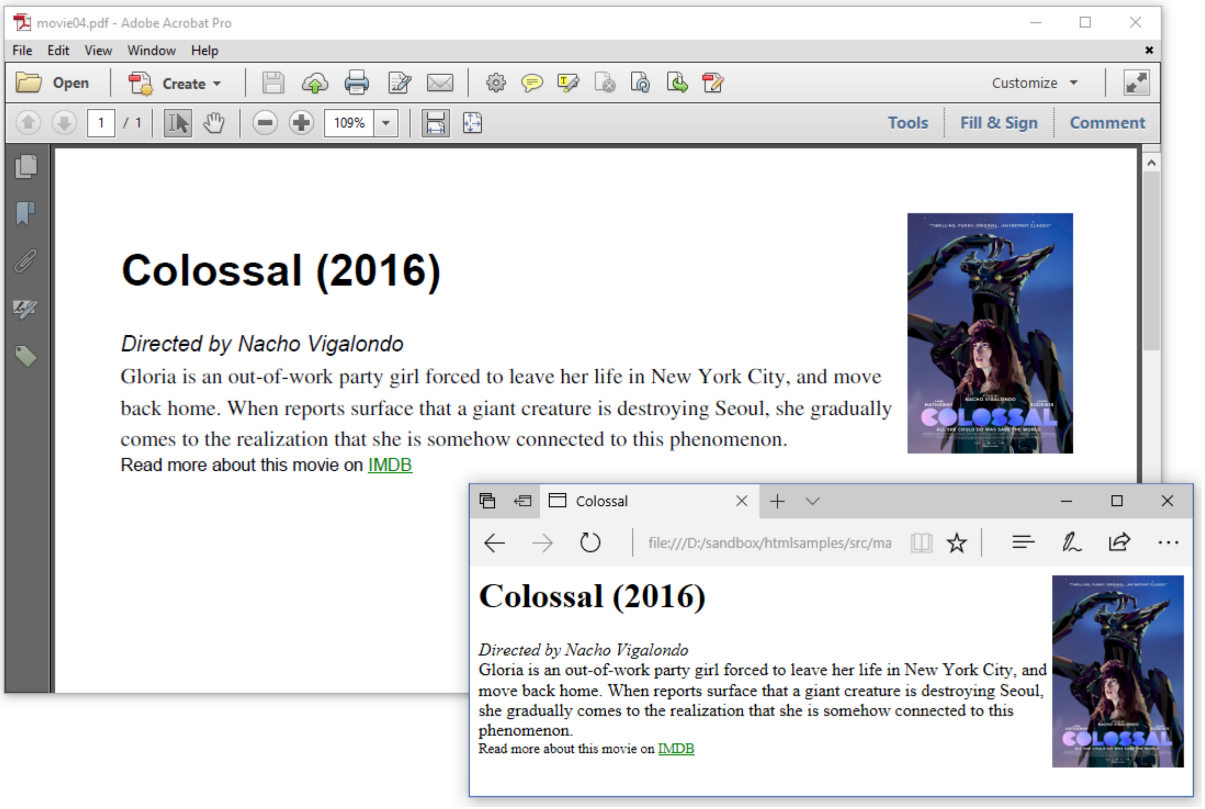
In 4_external_css.html, we don't have a `<style>` block in the header section. Instead we refer to a style sheet named movie.css using the `<link>` tag.

```
<html>
    <head>
        <title>Colossal</title>
        <meta name="description" content="Gloria is an out-of-work party girl ..." />
        <link rel="stylesheet" type="text/css" href="css/movie.css">
    </head>
    <body>
        <img src="img/colossal.jpg" class="poster" />
        <h1>Colossal (2016)</h1>
        <div class="director">Directed by Nacho Vigalondo</div>
        <div class="description">
        Gloria is an out-of-work party girl forced to leave her life in New York City,
        and move  back home. When reports surface that a giant creature is
        destroying Seoul, she gradually comes to the realization that she is
        somehow connected to this phenomenon.
        </div>
        <div class="imdb">Read more about this movie on
        <a href="www.imdb.com/title/tt4680182">IMDB</a></div>
    </body>
</html>
```

The external CSS file, movie.css, looks like this:

```
.poster {
    width: 120px;
    float: right;
}
.director {
    font-style: italic;
}
.description {
    font-family: serif;
}
.imdb {
    font-size: 0.8em;
}
a {
    color: green;
}
```

Apart from the fact that we changed the color of the content of the `<a>`-tag to green, there isn't much difference between figure 2.4 and figure 2.3.


movie04.pdf - Adobe Acrobat Pro
File Edit View Window Help
Open
Create
Customize
Tools
Fill & Sign
Comment
109%
Colossal
file:///D:/sandbox/htmlsamples/src/ma

Colossal (2016)

Directed by Nacho Vigalondo

Gloria is an out-of-work party girl forced to leave her life in New York City, and move back home. When reports surface that a giant creature is destroying Seoul, she gradually comes to the realization that she is somehow connected to this phenomenon.

Read more about this movie on IMDB

Figure 2.4: HTML page with external CSS

You can mix and match inline CSS, CSS in the header, and external CSS. You can even define a style in one place that is overridden in an other place.

When CSS is defined in different places, which CSS takes precedence?

CSS stands for *Cascading* Style Sheets, and the styles defined on difference levels “cascade” into a new, virtual style sheet, combining the styles by the following rules:

1. First there is the style sheet used by the browser. This style sheet is used in absence of specific styles. The pdfHTML add-on ships with its own stylesheet, which explains for instance why the content of an `<a>` tag is rendered in blue.
2. The style sheet used by the browser can be overruled by an external style sheet.
3. All the previously defined styles can be overruled by an internal style sheet that is present in the header section of the HTML file.
4. Inline styles (inside an HTML element) have top priority. When a style is defined in a `style` attribute, it overrules the previously defined style attributes.

In these examples, we positioned the image by creating a class named `poster`. We used CSS to define the width, and to position the image to the right with `float`. In the next example, we are going to use absolute positions.

Using absolute positioning

In posters.html, we use a distance from the top and the left side of the page to define an absolute position.

```
<html>
    <head><title>SXSW movies</title></head>
    <body>
        <img src="img/68_kill.jpg"
            style="position: absolute; top: 5; left: 5; width: 60;">
        <img src="img/a_bad_idea_gone_wrong.jpg"
            style="position: absolute; top: 5; left: 70; width: 60;">
        <img src="img/a_critically_endangered_species.jpg"
            style="position: absolute; top: 5; left: 135; width: 60;">
        <img src="img/big_sick.jpg"
            style="position: absolute; top: 5; left: 200; width: 60;">
        <img src="img/california_dreams.jpg"
            style="position: absolute; top: 5; left: 265; width: 60;">
        <img src="img/colossal.jpg"
            style="position: absolute; top: 5; left: 330; width: 60;">
        <img src="img/daraju.jpg"
            style="position: absolute; top: 5; left: 395; width: 60;">
        <img src="img/drib.jpg"
            style="position: absolute; top: 5; left: 460; width: 60;">
        <img src="img/hot_summer_nights.jpg"
            style="position: absolute; top: 5; left: 525; width: 60;">
        <img src="img/unrest.jpg"
            style="position: absolute; top: 5; left: 590; width: 60;">
        <img src="img/hounds_of_love.jpg"
            style="position: absolute; top: 120; left: 5; width: 60;">
        <img src="img/lane1974.jpg"
            style="position: absolute; top: 120; left: 70; width: 60;">
        <img src="img/madre.jpg"
            style="position: absolute; top: 120; left: 135; width: 60;">
        <img src="img/mfa.jpg"
            style="position: absolute; top: 120; left: 200; width: 60;">
        <img src="img/mr_roosevelt.jpg"
            style="position: absolute; top: 120; left: 265; width: 60;">
        <img src="img/nobody_speak.jpg"
            style="position: absolute; top: 120; left: 330; width: 60;">
        <img src="img/prevenge.jpg"
            style="position: absolute; top: 120; left: 395; width: 60;">
        <img src="img/the_archer.jpg"
            style="position: absolute; top: 120; left: 460; width: 60;">
        <img src="img/the_most_hated_woman_in_america.jpg"
            style="position: absolute; top: 120; left: 525; width: 60;">
        <img src="img/this_is_your_death.jpg"
            style="position: absolute; top: 120; left: 590; width: 60;">
    </body>
</html>
```

Figure 2.5 shows the result.

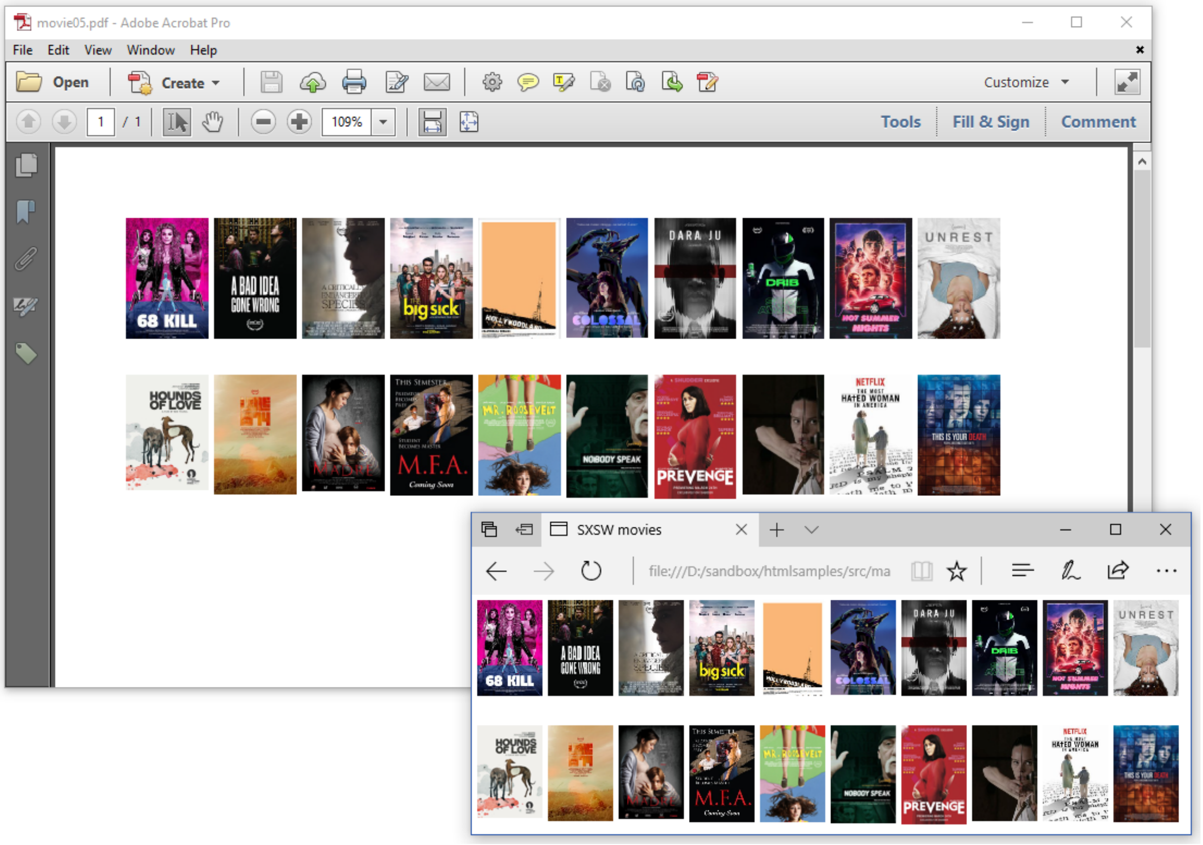

movie05.pdf - Adobe Acrobat Pro
File Edit View Window Help
Open
Create
Customize
1 / 1
109%
Tools
Fill & Sign
Comment
SXSW movies
file:///D:/sandbox/htmlsamples/src/ma

Figure 2.5: HTML page with images at absolute positions

Be careful when you use this functionality. The position you choose for an HTML page won't always fit a PDF page. You will have to take that into account, either when you design your HTML, or when you define the default page size of your PDF document.

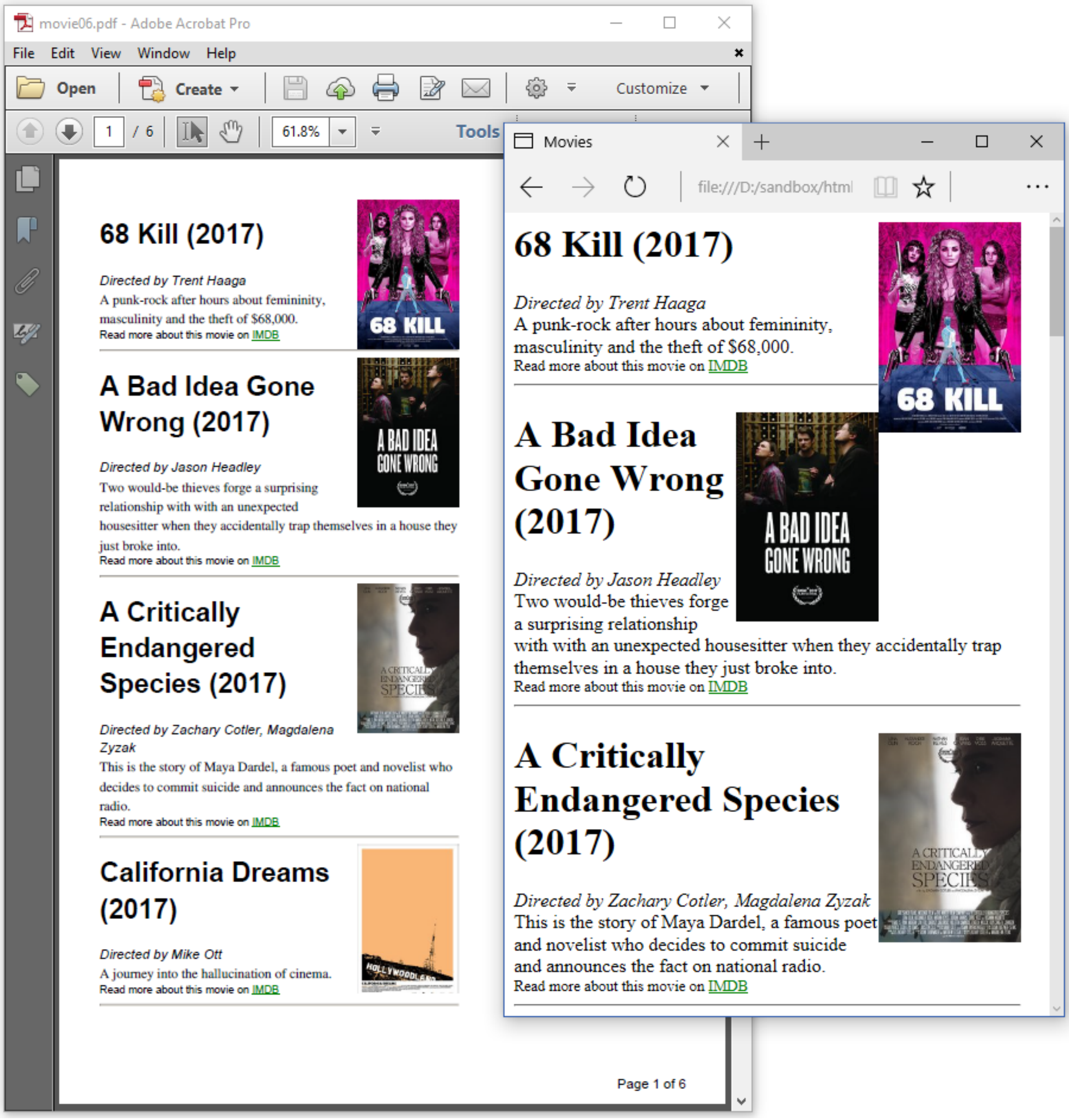
pdfHTML also supports at-rules.

Adding "Page X of Y" using an @page rule

In movies.html, we have a list of 20 movies that are presented using movie.css for the styles. Additionaly, we also defined a CSS at-rule in the header of the page.

```
<html>
    <head>
        <title>Movies</title>
        <meta name="description" content="Selection of movies screened at SXSW 2017" />
        <link rel="stylesheet" type="text/css" href="css/movie.css">
        <style>
            @page {
                @bottom-right {
                    content: "Page " counter(page) " of " counter(pages);
                }
            }
        </style>
    </head>
    <body>
        <div style="width: 320pt;">
            <img src="img/68_kill.jpg" class="poster" />
            <h1>68 Kill (2017)</h1>
            <div class="director">Directed by Trent Haaga</div>
            <div class="description">
                 A punk-rock after hours about femininity,
                 masculinity and the theft of $68,000.
            </div>
            <div class="imdb">Read more about this movie on
            <a href="www.imdb.com/title/tt5189894">IMDB</a></div>
            <hr>
        </div>
        <div style="width: 320pt;">
            <img src="img/a_bad_idea_gone_wrong.jpg" class="poster" />
            <h1>A Bad Idea Gone Wrong (2017)</h1>
            <div class="director">Directed by Jason Headley</div>
            <div class="description">
                Two would-be thieves forge a surprising relationship with with an
                unexpected housesitter when they accidentally trap themselves in
                a house they just broke into.
            </div>
            <div class="imdb">Read more about this movie on
            <a href="www.imdb.com/title/tt5212918">IMDB</a></div>
            <hr>
        </div>
        ...
    </body>
</html>
```

With `@page`, we define a footer –positioned `@bottom-right`– of which the `content` is composed as "“Page X of Y” where X is the value of the current page number, and Y is the total number of pages. In figure 2.6, we see **page 1 of 6** in the bottom-right corner of the page.



68 Kill (2017)

Directed by Trent Haaga
A punk-rock after hours about femininity, masculinity and the theft of $68,000.
Read more about this movie on IMDB

A Bad Idea Gone Wrong (2017)

Directed by Jason Headley
Two would-be thieves forge a surprising relationship with with an unexpected housesitter when they accidentally trap themselves in a house they just broke into.
Read more about this movie on IMDB

A Critically Endangered Species (2017)

Directed by Zachary Cotler, Magdalena Zyzak
This is the story of Maya Dardel, a famous poet and novelist who decides to commit suicide and announces the fact on national radio.
Read more about this movie on IMDB

California Dreams (2017)

Directed by Mike Ott
A journey into the hallucination of cinema.
Read more about this movie on IMDB

Page 1 of 6

Figure 2.6: HTML page with "page X of Y" footers

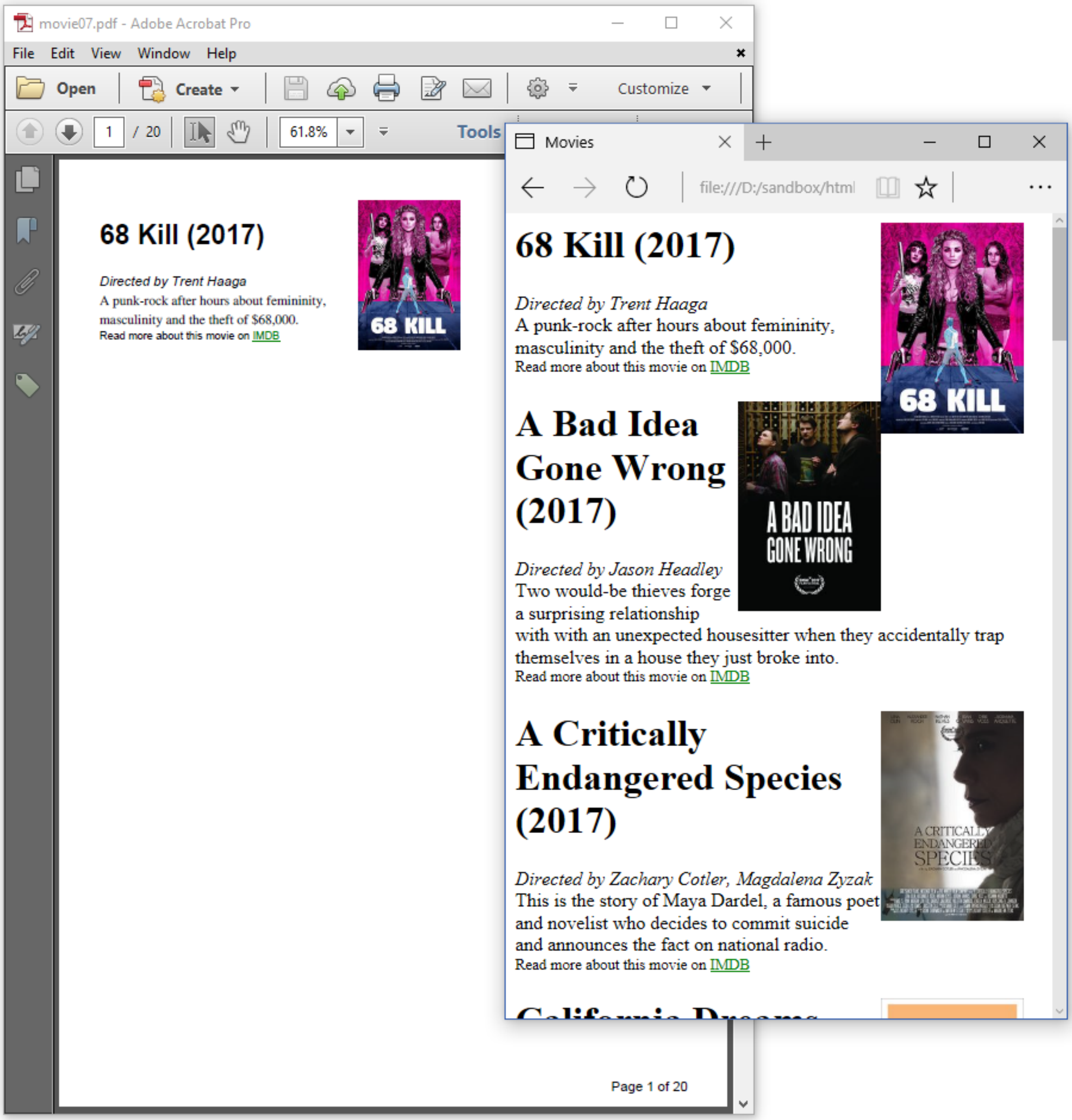
Now suppose that we want every movie to start on a new page. In that case, we'll introduce a page break.

Adding page breaks

We only made a single change in movies2.html. We added `page-break-after: always` to the top-`<div>` of every movie but the last:

```
<html>
    <head>
        <title>Movies</title>
        <meta name="description" content="Selection of SXSW movies" />
        <link rel="stylesheet" type="text/css" href="css/movie.css">
        <style>
            @page {
                @bottom-right {
                    content: "Page " counter(page) " of " counter(pages);
                }
            }
        </style>
    </head>
    <body>
        <div style="page-break-after: always; width: 320pt;">
            <img src="img/68_kill.jpg" class="poster" />
            <h1>68 Kill (2017)</h1>
            <div class="director">Directed by Trent Haaga</div>
            <div class="description">
                 A punk-rock after hours about femininity,
                 masculinity and the theft of $68,000.
            </div>
            <div class="imdb">Read more about this movie on
            <a href="www.imdb.com/title/tt5189894">IMDB</a></div>
            <hr>
        </div>
        ...
    </body>
</html>
```

The movie overview is now a document with 20 pages, one page per movie, as shown in figure 2.7.



68 Kill (2017)

Directed by Trent Haaga
A punk-rock after hours about femininity, masculinity and the theft of $68,000.
Read more about this movie on IMDB

Page 1 of 20

68 Kill (2017)

Directed by Trent Haaga
A punk-rock after hours about femininity, masculinity and the theft of $68,000.
Read more about this movie on IMDB

A Bad Idea Gone Wrong (2017)

Directed by Jason Headley
Two would-be thieves forge a surprising relationship with with an unexpected housesitter when they accidentally trap themselves in a house they just broke into.
Read more about this movie on IMDB

A Critically Endangered Species (2017)

Directed by Zachary Cotler, Magdalena Zyzak
This is the story of Maya Dardel, a famous poet and novelist who decides to commit suicide and announces the fact on national radio.
Read more about this movie on IMDB

Figure 2.7: HTML page with page breaks

There's much more that we could do with CSS, but let's see what we've done so far.

Summary

In this chapter, we've learned about inline CSS, CSS in the header, and external CSS. In the last example, we've even made a combination of inline CSS to force a page break, internal CSS in the header to add a footer, and external CSS for fonts, colors, and so on. In the next chapter, we're going to use some more CSS, more specifically in the context of media queries.