Scryber Core

Sep 01, 2021

Getting started

What if there was a better way to create documents, is at the heart of everything we do.

Scryber is **the** engine to create dynamic PDF documents quickly and easily from HTML templates with consistant styles, your own data, and an easy flowing layout. It's open source; flexible; styles based; data driven and with a low learning curve.

Written entirely in C# for dotnet 5 using HTML, CSS and SVG.

Documentation for previous 1.0.x pdfx versions for Read the docs here

CHAPTER 1

Hello World MVC

Download the nuget package

https://www.nuget.org/packages/Scryber.Core.Mvc

Start with a template. **The namespace declaration is important.**

```
<!DOCTYPE HTML >
<html lang='en' xmlns='http://www.w3.org/1999/xhtml' >
    <head>
        <title>{@:model}</title>
    </head>
    <body>
        <div style='padding:10px'>{@:model}.</div>
    </body>
</html>
```

And then generate your template in a view.

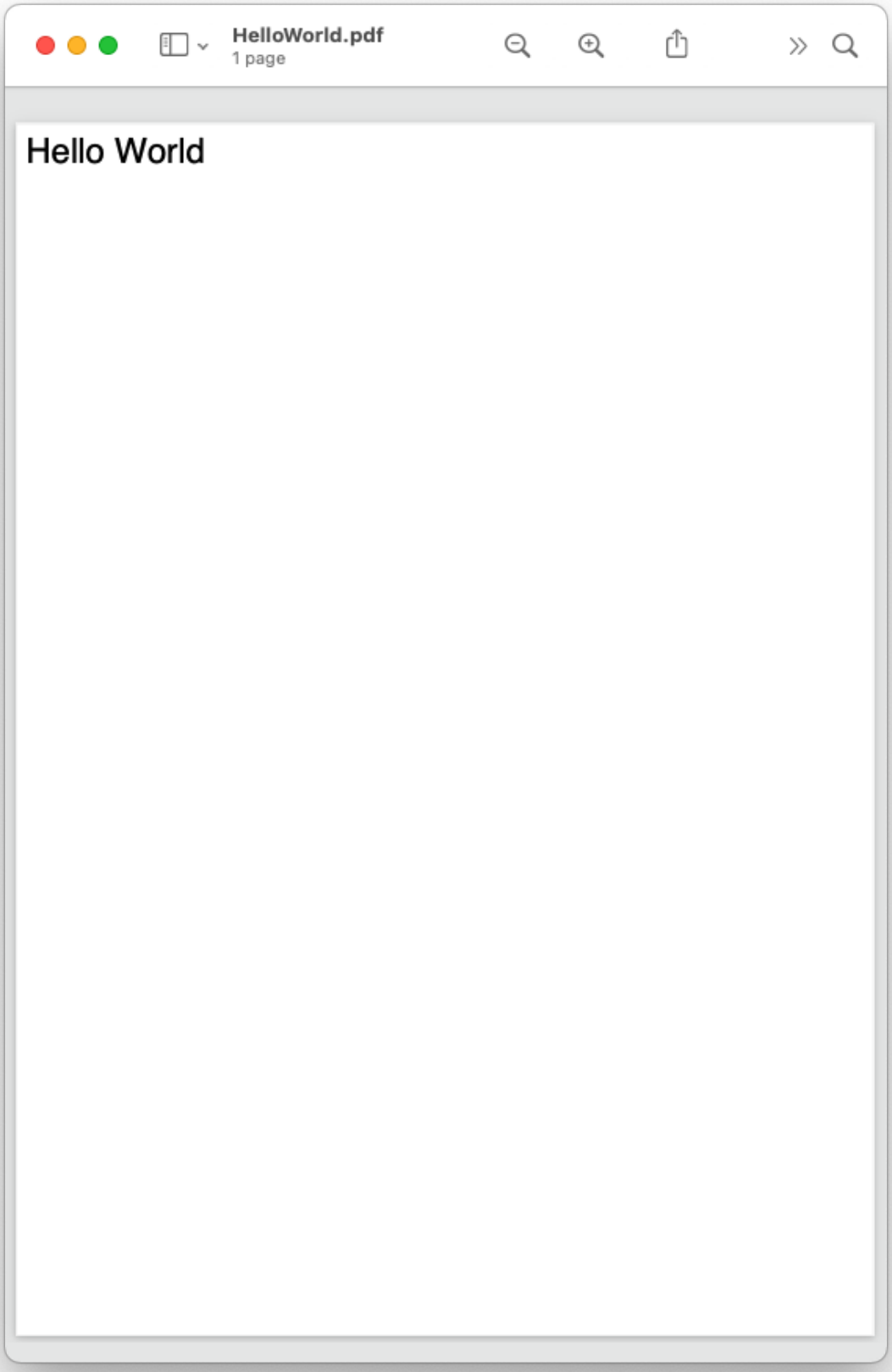
```
//add the namespaces
//using Scryber.Components;
//using Scryber.Components.Mvc;
//using Microsoft.AspNetCore.Mvc;

public class HomeController : Controller
{

    private readonly IWebHostEnvironment _env;

    public HomeController(IWebHostEnvironment environment)
    {
        _env = environment;
    }

    [HttpGet]
    public IActionResult HelloWorld()
```

(continues on next page)

(continued from previous page)

```
    {
        // get the path to where you have saved your template
        var path = _env.ContentRootPath;
        path = System.IO.Path.Combine(path, "Views", "PDF", "HelloWorld.html");

        //parsing the document creates a complete object graph from the content
        using(var doc = Document.ParseDocument(path))
        {
            doc.Params["model"] = "Hello World";
            return this.PDF(doc); //convenience extension method to return the result.
        }
    }

}
```

Try it out here…


HelloWorld.pdf
1 page
Hello World

CHAPTER 2

Hello World Plus

Check out *MVC Controller - Getting Started* for a full MVC example with styles and binding, or *Console or GUI - Getting Started* for a full gui application example (with styles and binding)

CHAPTER 3

Features

CHAPTER 4

Easy, and intuitive structure

Whether you are using xhtml templates or directly in code, scryber is quick and easy to build complex documents from your designs and data using standard xhtml.

CHAPTER 5

Intelligent flowing layout engine

In scryber, content can either be laid out explicitly, or jut flowing with the the page. Change the page size, or insert content and everything will adjust around it.

CHAPTER 6

Cascading Styles

With a styles based structure, it's easy to apply designs to templates. Use class names, id's or component types, or nested selectors.

CHAPTER 7

Drawing and Typographic support

Scryber supports inclusion of Images, Fonts (inc. Google fonts) and SVG components for drawing graphics and icons.

CHAPTER 8

Binding to your data

With a simple binding notation it's easy to add references to your data structures and pass information and complex data to your document from SQL, JSON, Entity Model and more. Or get the document to look up and bind the data for you.

CHAPTER 9

Extensible Framework

Scryber was designed from the ground up to be extensible. If it doesn't do what you need, then we think you can make it do it. With iFrame includes, a namespace based parser engine, and configuration options for images, fonts, binding it's down to your imagination

CHAPTER 10

Secure and Encrypted

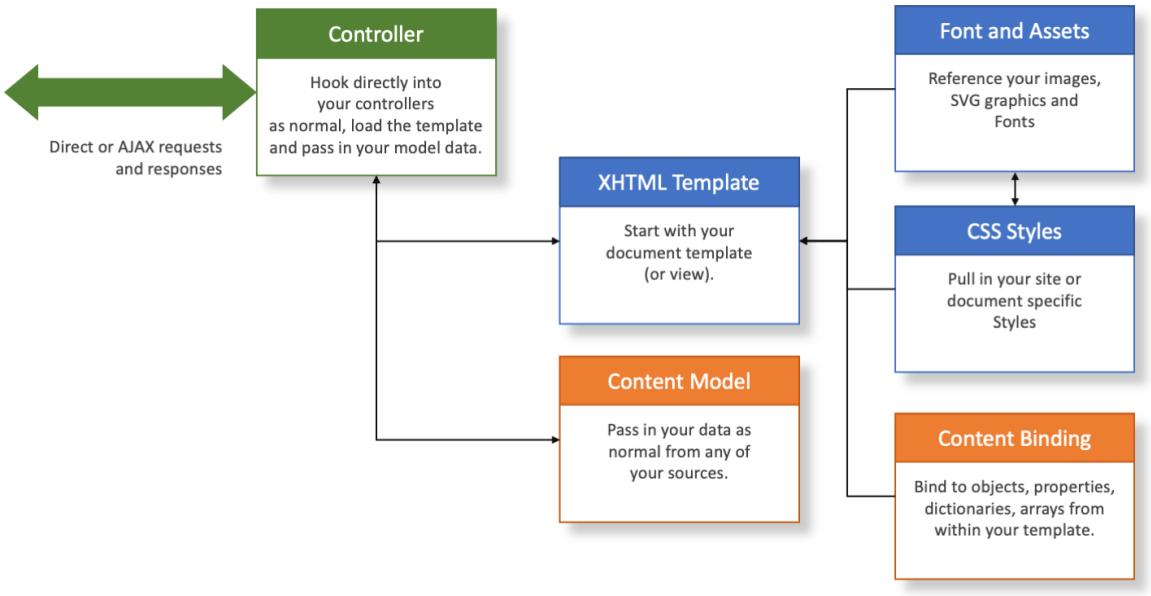
Scryber fully supports the PDF restrictions and both 40 bit and 128 bit encryption of documents using owner and user passwords.

10.1 MVC Controller - Getting Started

A Complete example for creating a styled and databound hello world PDF file from an MVC Controller in C# with an HTML template

10.1.1 How it works

We hope scryber works just as you would expect. The engine is based around the controllers you have, using HTML template views with css, graphics and images you are used to, along with your model data you have, to create PDF documents quickly, easily and flexibly.



10.1.2 Nuget Packages

If you have not done so already, make sure you install the Nuget Package in your new or existing MVC project.

https://www.nuget.org/packages/Scryber.Core.Mvc

This will add the latest version of the Scryber.Core, and also the Scryber.Core.Mvc Controller extension methods.

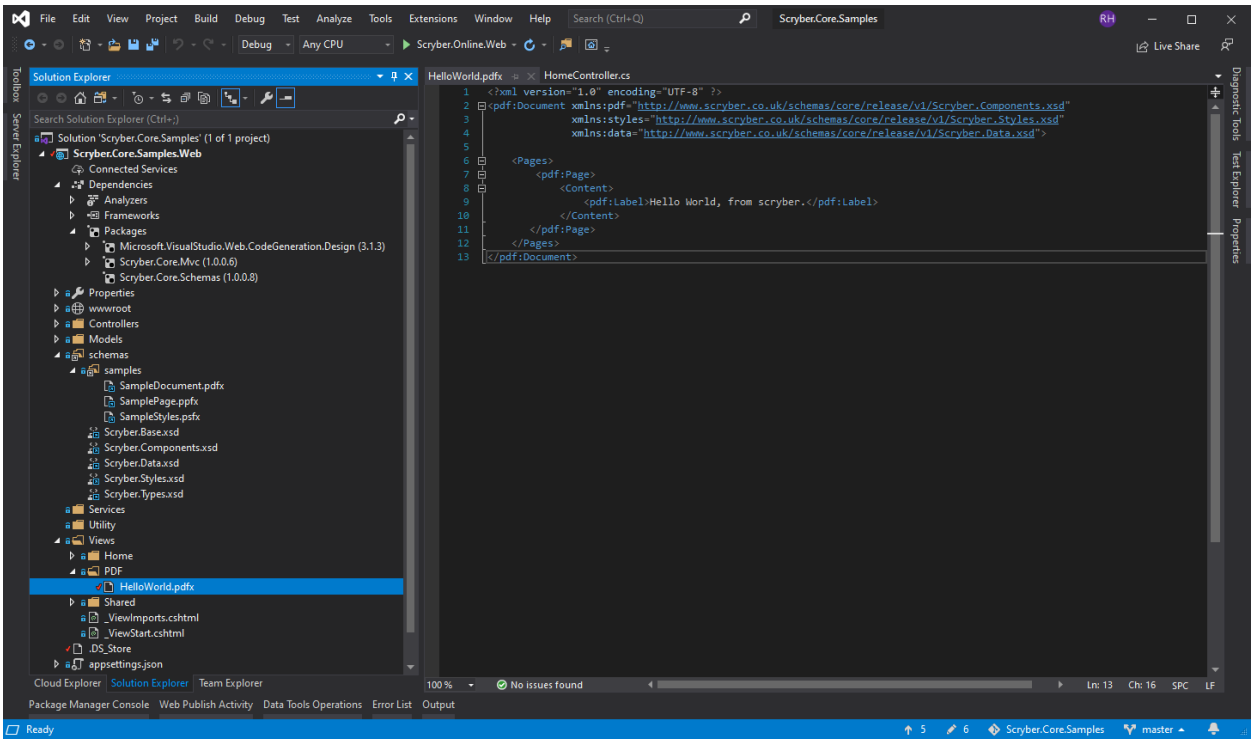
10.1.3 Add a document template

In our applications we like to add our templates to a PDF folder the Views folder. You can break it down however works for you, but for now, a create a new html file called HelloWorld.html in your folder.

And paste the following content into the file

```
<!DOCTYPE HTML >
<html lang='en' xmlns='http://www.w3.org/1999/xhtml' >
    <head>
        <title>Hello World</title>
    </head>
    <body>
        <div style='padding:10px'>Hello World from scryber.</div>
    </body>
</html>
```

Your solution should look something like this.



The xmlns is really important for knowing what type of document and schema is being used.

For for more information on the namespaces and mappings see this *Namespaces and their Assemblies* documentation

10.1.4 Controller code

Add a new controller to your project, and a couple of namespaces are important to add to the top of your controller.

```
using Scryber.Components;
using Scryber.Components.Mvc;
```

10.1.5 Add the Web host service

In order to nicely reference files in the solution, we add a reference to the IWebHostEnvironment to the home controller constructor.

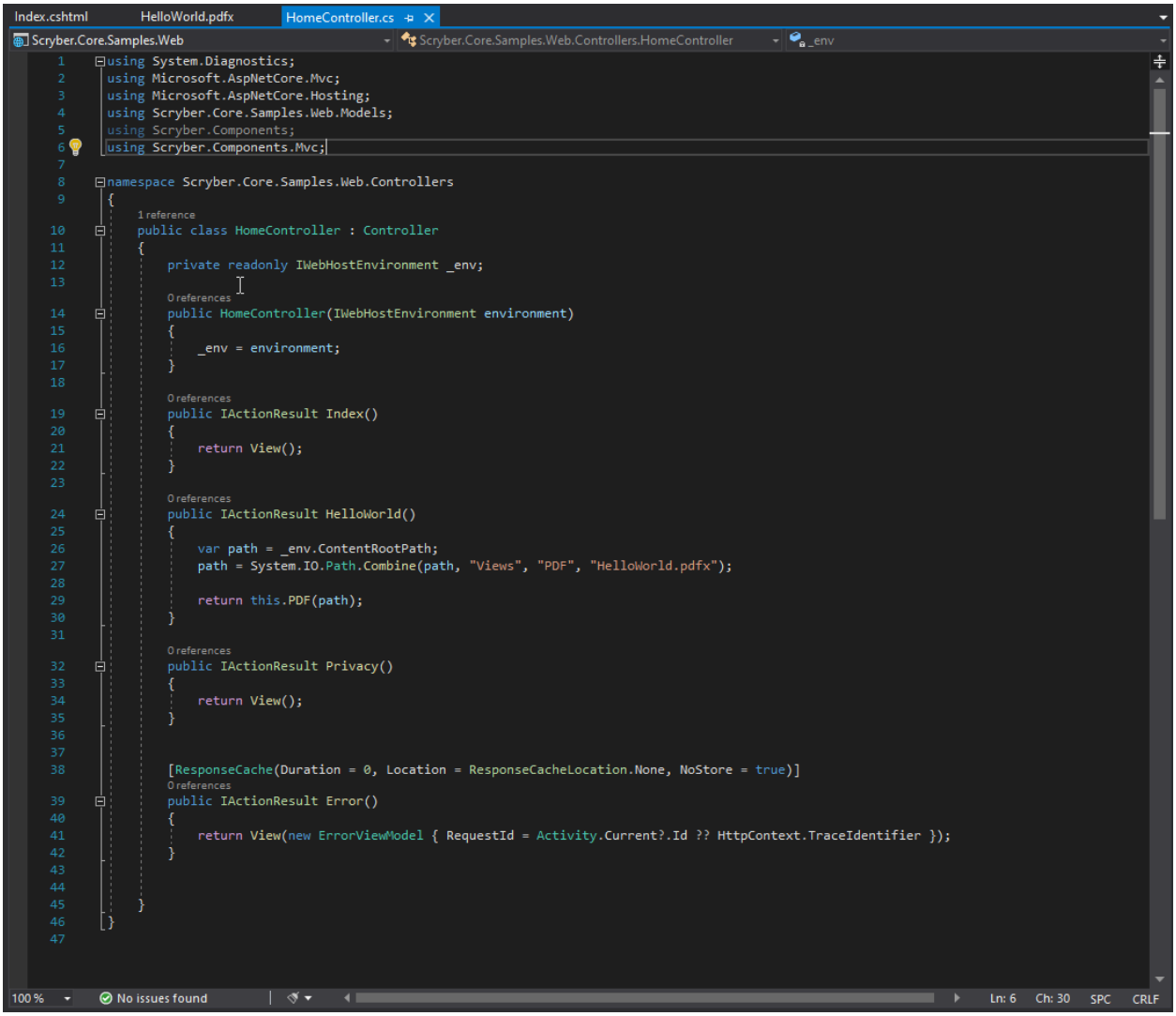
```
private readonly IWebHostEnvironment _env;

public HomeController(IWebHostEnvironment environment)
{
    _env = environment;
}
```

10.1.6 Add a Controller Method

Next add a new Controller Method to your class for retrieve and generate

```
[HttpGet]
public IActionResult HelloWorld()
{
    var path = _env.ContentRootPath;
    path = System.IO.Path.Combine(path, "Views", "PDF", "HelloWorld.pdfx");

    using(var doc = Document.ParseDocument(path))
        return this.PDF(doc);
}
```

The PDF extension method will read the PDF template from the path and generate the file to the response.



10.1.7 Testing your action

To create your pdf simply add a link to your action method in a view.

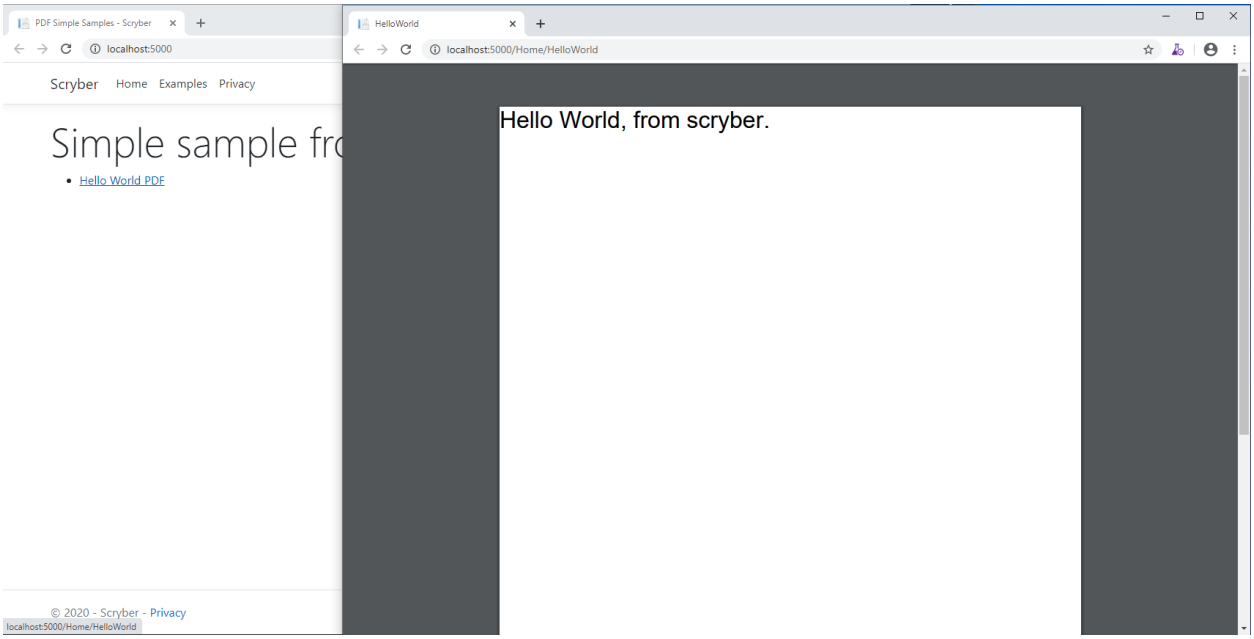
```
<div>
    <h2 class="display-4">Simple sample from the PDF Controller</h2>
```

(continues on next page)

(continued from previous page)

```
    <ul>
        <li><a href='@Url.Action("HelloWorld","Home")' target='_blank'>Hello World PDF
↪</a></li>
    </ul>
</div>
```

Running your application, you should see the link and clicking on it will open the pdf in a new tab or window.



10.1.8 Adding dynamic content

One of the driving forces behind scryber is the separation of the content, data and style. It is common practice in sites. With scryber all attributes and content is bindable to the data you want to pass to it,

So we can specify our model data with from any source (here we are just using a dynamic object). And we can pass it to the parsed document either explicitly, or using the special 'model' overload on the PDF extension method.

```
private dynamic GetHelloWorldData()
{
    //get your model data however you wish
    //it's just a sample object for this one.

    var model = new
        {
            titlestyle = "color:#ff6347", //style data
            title = "Hello from scryber", //simple content
            author = "Joe the Mack",
            items = new[]                 //or even complex object data
            {
                new { name = "First item" },
                new { name = "Second item" },
                new { name = "Third item" },
            }
        };
```

(continues on next page)

(continued from previous page)

```
    return model;
}

[HttpGet]
public IActionResult HelloWorld()
{
    var path = _env.ContentRootPath;
    path = System.IO.Path.Combine(path, "Views", "PDF", "HelloWorld.html");

    using(var doc = Document.ParseDocument(path))
    {
        var model = GetHelloWorldData();

        //could use doc.Params["model"] = model; for the same effect.
        //It is just more convenient as below.
        return this.PDF(doc, model);
    }
}
```

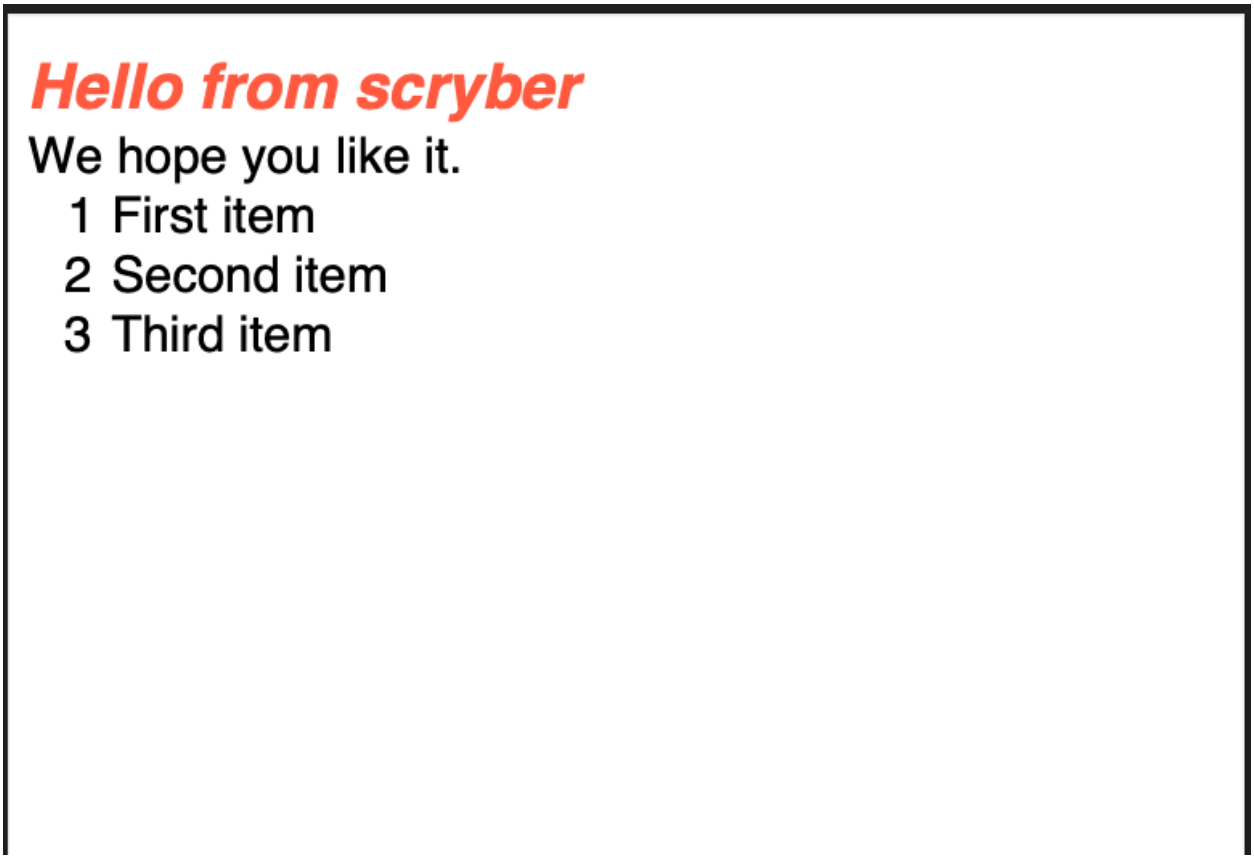
The general syntax for referring paramters in a template is

{@:**parameter[.property]**}

And the html5 tag 'template' is used with the data-bind attribute to loop over one or more items in a collection, and the inner objects and properties can be used with the '.' prefix to reference the current data context.

So we can expand our document body to use the model schema.

```
<body>

    <main style="padding:10pt">

        <!-- binding styles and values on content -->
        <h2 style="{@:model.titlestyle}">{@:model.title}</h2>

        <div>We hope you like it.</div>

        <!-- Loop with nested item collection binding to the objects -->
        <ol>
            <template data-bind='{@:model.items}'>
                <!-- binding within the model.items content, and can be nested -->
                <li>{@:.name}</li>
            </template>
        </ol>
    </main>

</body>
```



10.1.9 Adding Fonts and Styles

It's good but rather uninspiring. With scryber we can use css styles, just as we would in html.

Here we are:

- Adding a stylesheet link to the google 'Fraunces' font with the @font-face at-rule (watch that &display=swap link - it's not xhtml)
- Adding some document styles for the body with fall-back fonts.
- A complex style for a page header, with a colour and single background image, that will be repeated across any page.
- And a page footer table with full width and associated style on the inner cells, that will again be repeated.

The css style could just have easily come from another referenced stylesheet.

```
<!DOCTYPE HTML >
<html lang='en' xmlns='http://www.w3.org/1999/xhtml' >
    <head>
        <title>Hello World</title>

        <!-- support for complex css selectors (or link ot external style sheets )-->
        <link rel="stylesheet"
            href="https://fonts.googleapis.com/css2?family=Fraunces:ital,wght@0,400;0,
↪700;1,400;1,700&amp;display=swap"
            title="Fraunces" />
```

(continues on next page)

(continued from previous page)

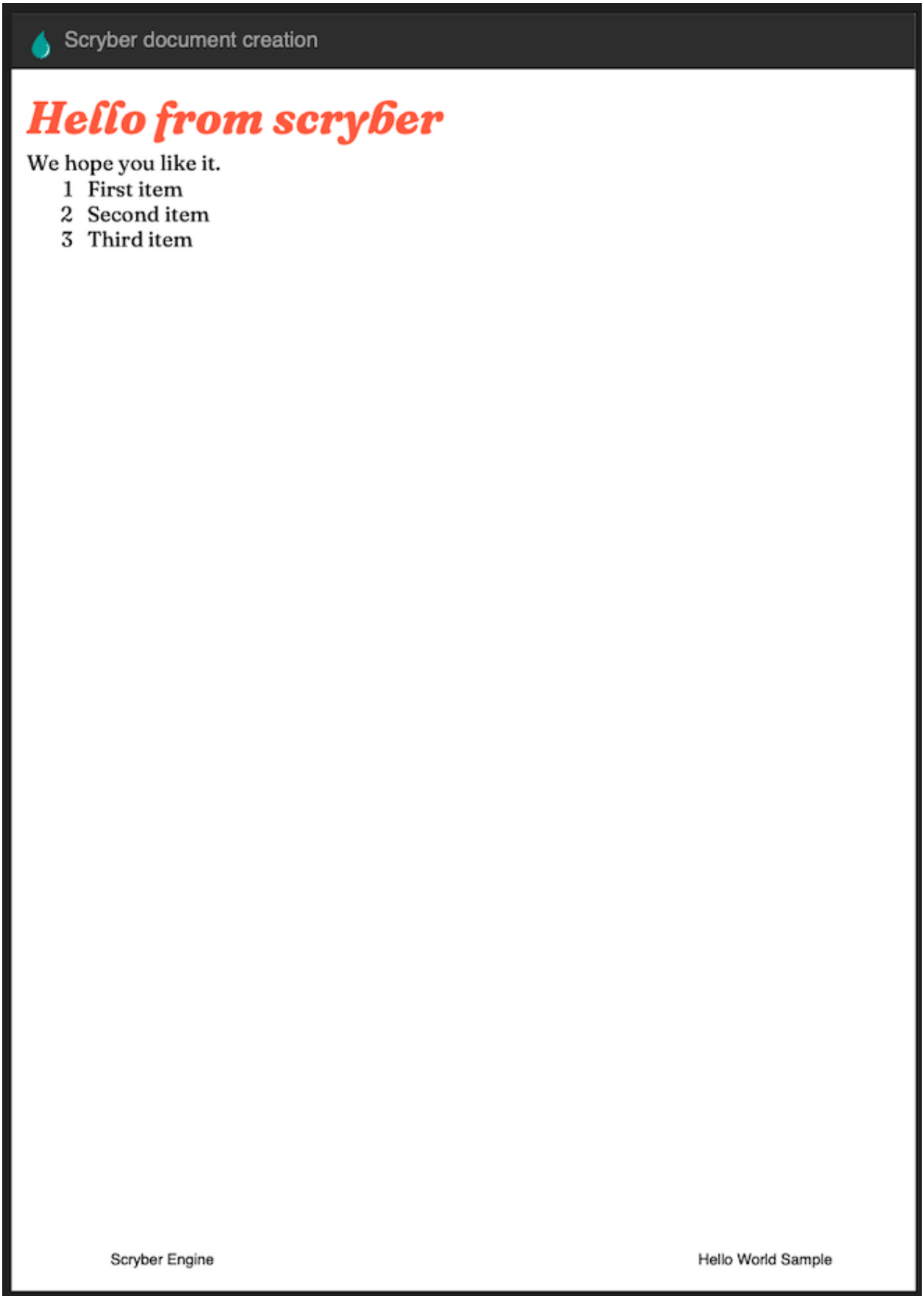
```
        <style>
            body {
                font-family: 'Fraunces', sans-serif;
                font-size: 14pt;
            }

            p.header {
                color: #AAA;
                background-color: #333;
                background-image: url('./html/images/ScyberLogo2_alpha_small.png');
                background-repeat: no-repeat;
                background-position: 10pt 10pt;
                background-size: 20pt 20pt;
                margin-top: 0pt;
                padding: 10pt 10pt 10pt 35pt;
            }


            .foot td {
                border: none;
                text-align: center;
                font-size: 10pt;
                margin-bottom: 10pt;
            }
        </style>
        <!-- Setting the base url for the references so they load the style sheet
↪background image from git -->
        <base href='https://raw.githubusercontent.com/richard-scryber/scryber.core/
↪master/Scryber.Core.UnitTest/Content/' />
    </head>
    <body>
        <header>
            <!-- document headers -->
            <p class="header">Scryber document creation</p>
        </header>
        <!-- support for many HTML5 tags-->
        <main style="padding:10pt">

            <!-- binding style and values on content -->
            <h2 style="{@:model.titlestyle}">{@:model.title}</h2>
            <div>We hope you like it.</div>
            <ol>
                <!-- Loop through the items in the model -->
                <template data-bind='{@:model.items}'>
                    <li>{@:.name}</li> <!-- and bind the name value -->
                </template>
            </ol>
        </main>
        <footer>
            <!-- footers in a table with style -->
            <table class="foot" style="width:100%">
                <tr>
                    <td>{@:model.author}</td>
                    <td>Hello World Sample</td>
                </tr>
            </table>
        </footer>
```

(continues on next page)

(continued from previous page)

```
    </body>
</html>
```

The output from this is much more pleasing. Especially that Fruances font :-)


Scryber document creation
Hello from scryber
We hope you like it.
1 First item
2 Second item
3 Third item
Scryber Engine
Hello World Sample

10.1.10 Page Breaks and sizes

Scryber's flexible layout engine supports the use of the @media and @page css at-rules to be able to explicitly style content just for document output, and also set page sizes and orientiations.

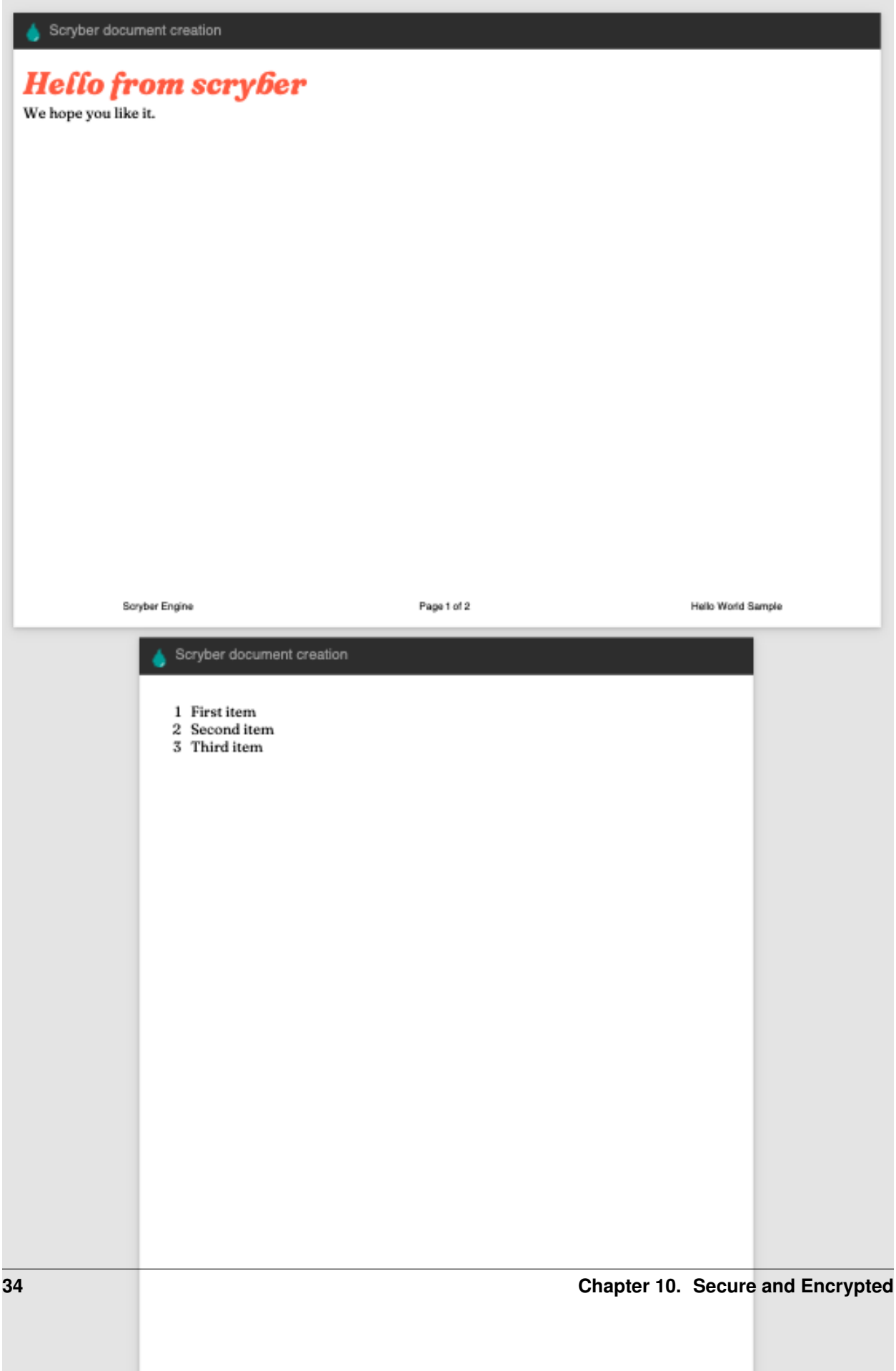
If we add a new css styles file to our site we can alter the layout easily and know it will not affect anything else we might be using our page for.

```
@media print {

    /* We have a new page before the ordered list
     * And this new page uses the 'portrait' option */
    main ol {
        page-break-before: always;
        padding-top: 20pt;
        page: portrait;
    }

    /* Default style is A4 landscape */

    @page {
        size: A4 landscape;
    }

    /* But can use specific sizes for sections */
    @page portrait {
        size: A4 portrait;
    }

}
```

In our footer we can add the current page number (of total pages) and an author property directly

```
<footer>
    <table class="foot" style="width:100%">
        <tr>
            <td>{@:model.author}</td>

            <!-- the page tag is made up, and has a property attribute
                (open to suggestions on better syntax)  -->

            <td>Page <page /> of <page property="total" /></td>
            <td>Hello World Sample</td>
        </tr>
    </table>
</footer>
```

With this we now alter the layout to use our new pages, and everything will flow nicely.


Scryber document creation
Hello from scryber
We hope you like it.
Scryber Engine
Page 1 of 2
Hello World Sample
Scryber document creation
1 First item
2 Second item
3 Third item

10.1.11 Further reading

You can read more about the what you can do with scryber from the contents.

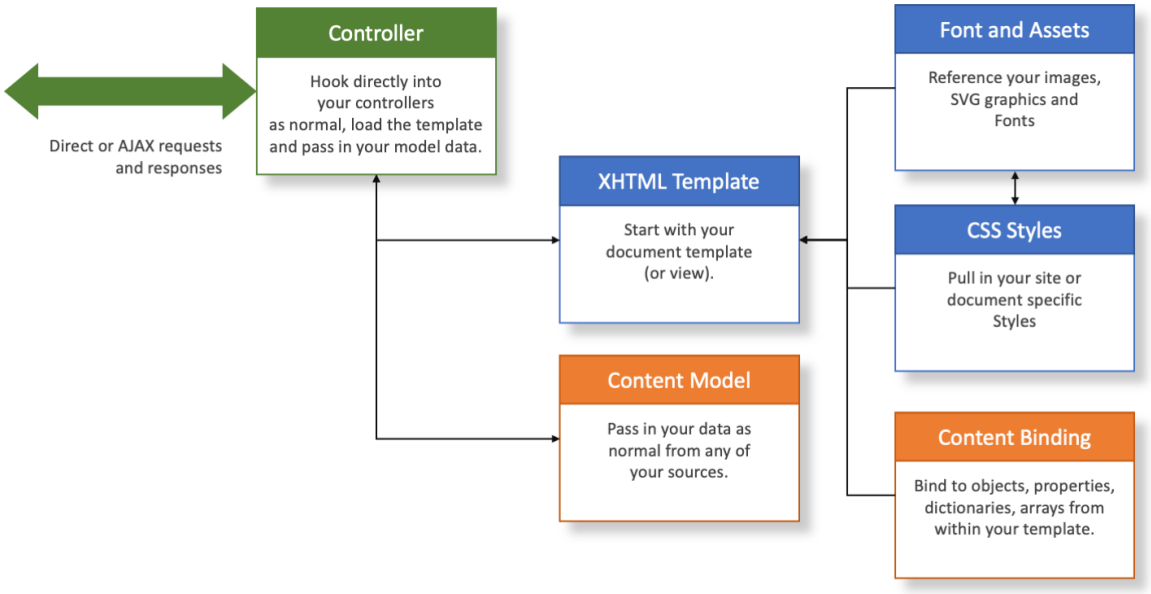
We have no idea what you will be able to create with scryber. It's just there to hopefully help you build amazing documents in an easy and repeatable way.

10.2 Console or GUI - Getting Started

A Complete example for creating a hello world PDF file in a console application or GUI front end. For us, we have just created a new dotnet core console application in Visual Studio.

10.2.1 How it works

We hope scryber works just as you would expect. The engine is based around the controllers you have, using XHTML template views with css, graphics and images you are used to, along with your model data you have, to create PDF documents quickly, easily and flexibly.



10.2.2 Nuget Packages

Make sure you install the Nuget Packages from the Nuget Package Manager

https://www.nuget.org/packages/Scryber.Core

This will add the latest version of the Scryber.Core nuget package.

10.2.3 Add a document template

In our applications we like to add our templates to a PDF folder. You can break it down however works for you, but for now a create a new XHML file called HelloWorld.html in your folder.

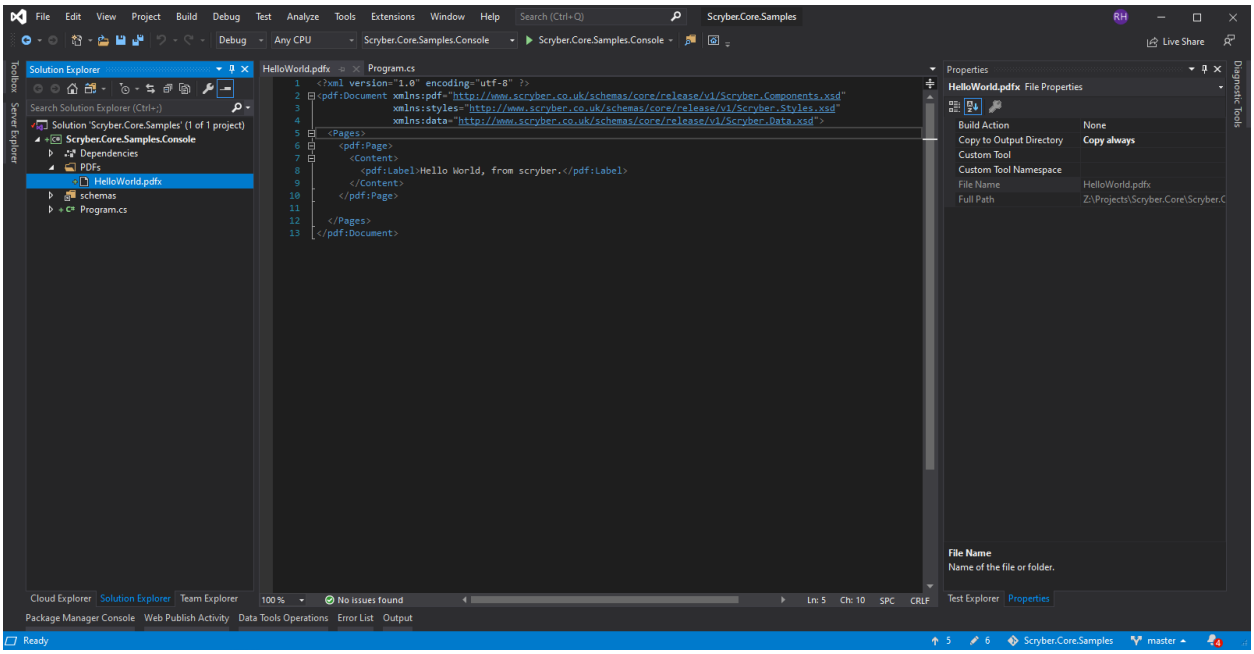
And paste the following content into the file

```
<!DOCTYPE HTML >
<html lang='en' xmlns='http://www.w3.org/1999/xhtml' >
    <head>
        <title>Hello World</title>
    </head>
    <body>
        <div style='padding:10px'>Hello World from scryber.</div>
    </body>
</html>
```

10.2.4 File properties

In the file properties for the HelloWorld.html file: Set the Build Action to None (if it is not already) And the Copy to output to Always.

Your solution should look something like this.



10.2.5 Program code

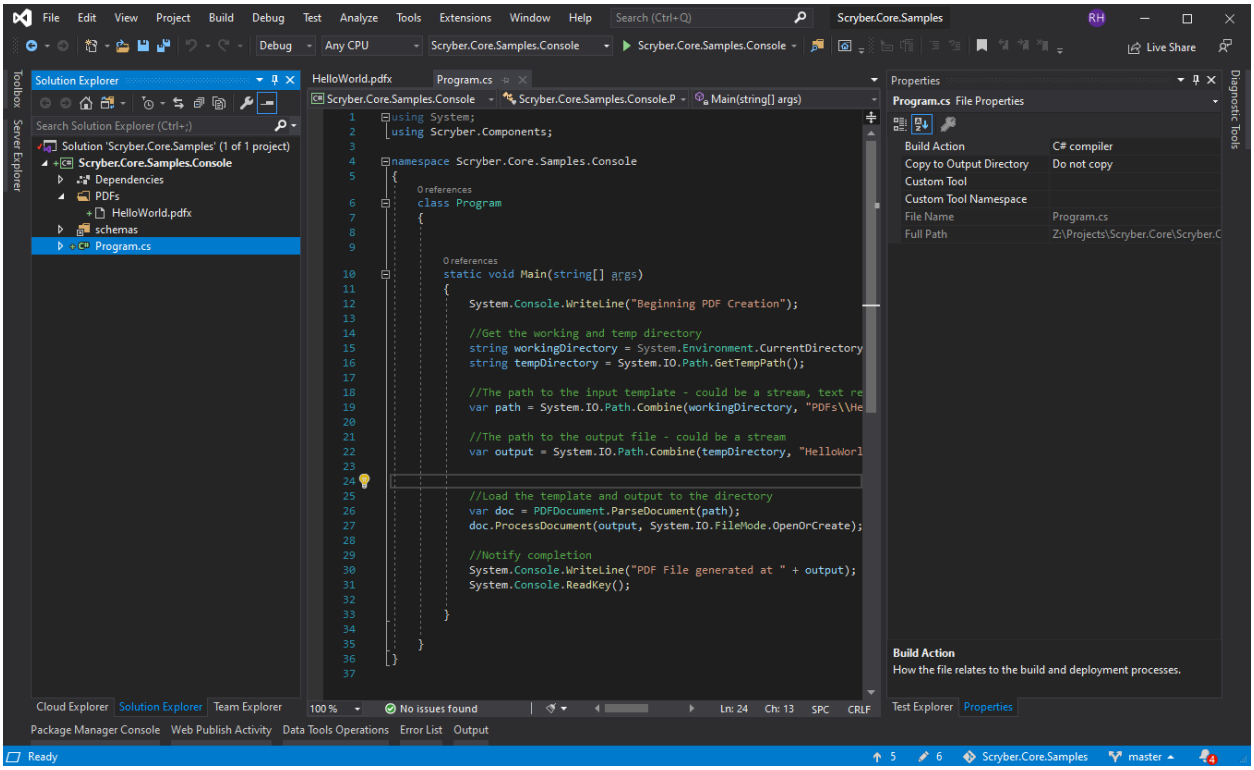
In your program.cs add the namespace to the top of your class.

```
using Scryber.Components;
```

10.2.6 Replace your main program method.

Next change the 'Main' method to your class to load the template and generate the pdf file

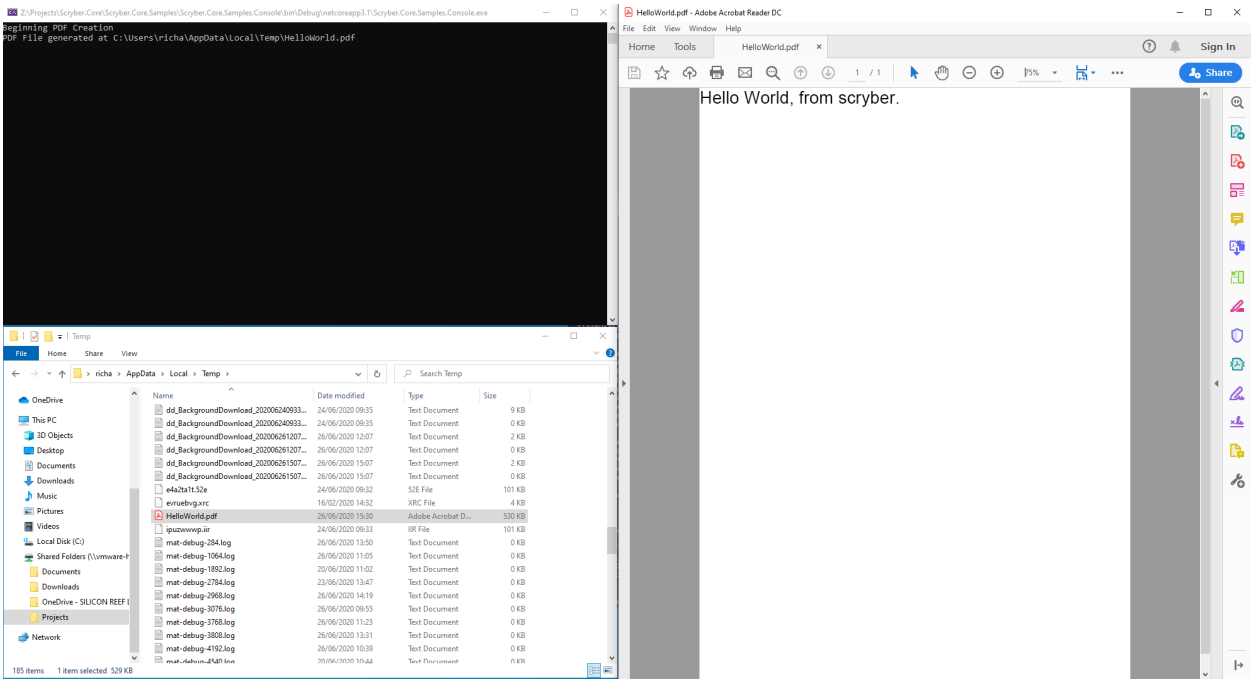
```
static void Main(string[] args)
{
    System.Console.WriteLine("Beginning PDF Creation");

    //Get the working and temp directory
    string workingDirectory = System.Environment.CurrentDirectory;
    string tempDirectory = System.IO.Path.GetTempPath();

    //The path to the input template - could be a stream, text reader, xml reader,␣
↪resource etc
    var path = System.IO.Path.Combine(workingDirectory, "PDFs\\HelloWorld.html");

    //The path to the output file - could be a stream
    var output = System.IO.Path.Combine(tempDirectory, "HelloWorld.pdf");


    //Load the template and output to the directory
    var doc = Document.ParseDocument(path);
    doc.SaveAsPDF(output, System.IO.FileMode.OpenOrCreate);

    //Notify completion
    System.Console.WriteLine("PDF File generated at " + output);
    System.Console.ReadKey();

}
```



The parser will read the document from the pdfx XML content, and then create a new PDF document in the tempDirectory for the output.

10.2.7 Testing your code

Running your application, you should see the console output the path to the pdf. And opening this will show you the file. you could have saved it to a share, opened in Acrobat reader, or sent via email as a stream attachment.



10.2.8 Adding dynamic content

One of the driving forces behind scryber is the separation of the content, data and style. It is common practice in sites. With scryber all attributes and content is bindable to the data you want to pass to it,

So we can specify our model data with from any source (here we are just using a dynamic object). And we can pass it to the parsed document either explicitly, or using the special 'model' overload on the PDF extension method.

```
private static dynamic GetHelloWorldData()
{
    //get your model data however you wish
    //it's just a sample object for this one.

    var model = new
        {
            titlestyle = "color:#ff6347", //style data
            title = "Hello from scryber", //simple content
            items = new[]                 //or even complex object data
            {
                new { name = "First item" },
                new { name = "Second item" },
                new { name = "Third item" },
            }
        };

    return model;
}
```

(continues on next page)

(continued from previous page)

```
static void Main(string[] args)
{
    System.Console.WriteLine("Beginning PDF Creation");

    string workingDirectory = System.Environment.CurrentDirectory;
    string tempDirectory = System.IO.Path.GetTempPath();

    var path = System.IO.Path.Combine(workingDirectory, "PDFs\\HelloWorld.html");

    var output = System.IO.Path.Combine(tempDirectory, "HelloWorld.pdf");

    var doc = Document.ParseDocument(path);

    //Assign the data model to a parameter
    doc.Params["model"] = GetHelloWorldData();

    doc.SaveAsPDF(output, System.IO.FileMode.OpenOrCreate);

    //Notify completion
    System.Console.WriteLine("PDF File generated at " + output);
    System.Console.ReadKey();

}
```

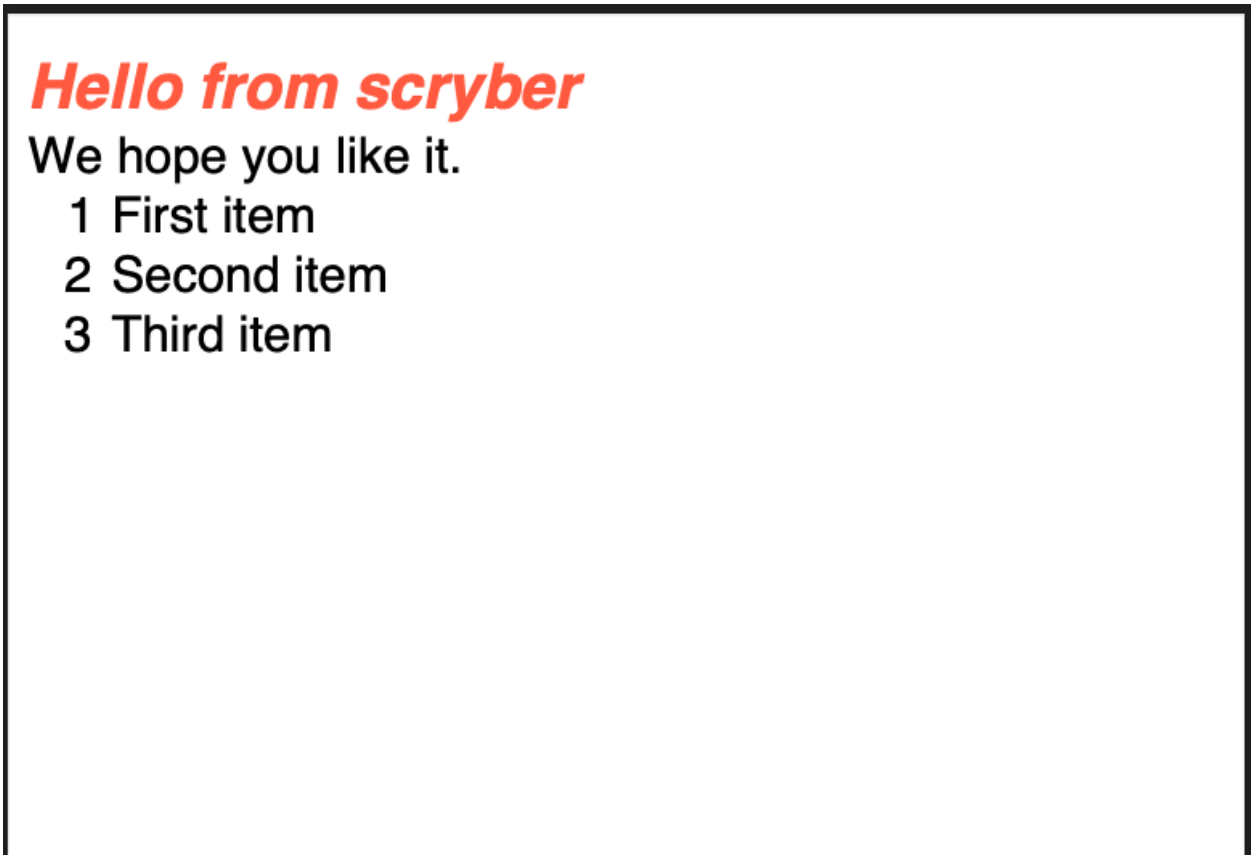
The general syntax for referring paramters in a template is

{@:**parameter[.property]**}

And the html5 tag 'template' is used with the data-bind attribute to loop over one or more items in a collection, and the inner objects and properties can be used with the '.' prefix to reference the current data context.

So we can expand our document body to use the model schema.

```
<body>

    <main style="padding:10pt">

        <!-- binding styles and values on content -->
        <h2 style="{@:model.titlestyle}">{@:model.title}</h2>

        <div>We hope you like it.</div>

        <!-- Loop with nested item collection binding to the objects -->
        <ol>
            <template data-bind='{@:model.items}'>
                <!-- binding within the model.items content, and can be nested -->
                <li>{@:.name}</li>
            </template>
        </ol>
    </main>

</body>
```



10.2.9 Adding Fonts and Styles

It's good but rather uninspiring. With scryber we can use css styles, just as we would in html.

Here we are:

- Adding a stylesheet link to the google 'Fraunces' font with the @font-face at-rule (watch that &display=swap link - it's not xhtml)
- Adding some document styles for the body with fall-back fonts.
- A complex style for a page header, with a colour and single background image, that will be repeated across any page.
- And a page footer table with full width and associated style on the inner cells, that will again be repeated.

The css style could just have easily come from another referenced stylesheet. Do not forget to encode the & character as &

```
<!DOCTYPE HTML >
<html lang='en' xmlns='http://www.w3.org/1999/xhtml' >
    <head>
        <title>Hello World</title>

        <!-- support for complex css selectors (or link ot external style sheets )-->
        <link rel="stylesheet"
            href="https://fonts.googleapis.com/css2?family=Fraunces:ital,wght@0,400;0,
↪700;1,400;1,700&amp;display=swap"
```

(continues on next page)

(continued from previous page)

```
            title="Fraunces" />

        <style>
            body {
                font-family: 'Roboto', sans-serif;
                font-size: 14pt;
            }

            p.header {
                color: #AAA;
                background-color: #333;
                background-image: url('https://avatars.githubusercontent.com/u/
↪65354830?s=64&amp;v=4');
                background-repeat: no-repeat;
                background-position: 10pt 10pt;
                background-size: 20pt 20pt;
                margin-top: 0pt;
                padding: 10pt 10pt 10pt 35pt;
            }


            .foot td {
                border: none;
                text-align: center;
                font-size: 10pt;
                margin-bottom: 10pt;
            }
        </style>
    </head>
    <body>
        <header>
            <!-- document headers -->
            <p class="header">Scryber document creation</p>
        </header>
        <!-- support for many HTML5 tags-->
        <main style="padding:10pt">

            <!-- binding style and values on content -->
            <h2 style="{@:model.titlestyle}">{@:model.title}</h2>
            <div>We hope you like it.</div>
            <ol>
                <!-- Loop through the items in the model -->
                <template data-bind='{@:model.items}'>
                    <li>{@:.name}</li> <!-- and bind the name value -->
                </template>
            </ol>
        </main>
        <footer>
            <!-- footers in a table with style -->
            <table class="foot" style="width:100%">
                <tr>
                    <td>{@:model.author}</td>
                    <td>Hello World Sample</td>
                </tr>
            </table>
        </footer>
    </body>
```

(continues on next page)

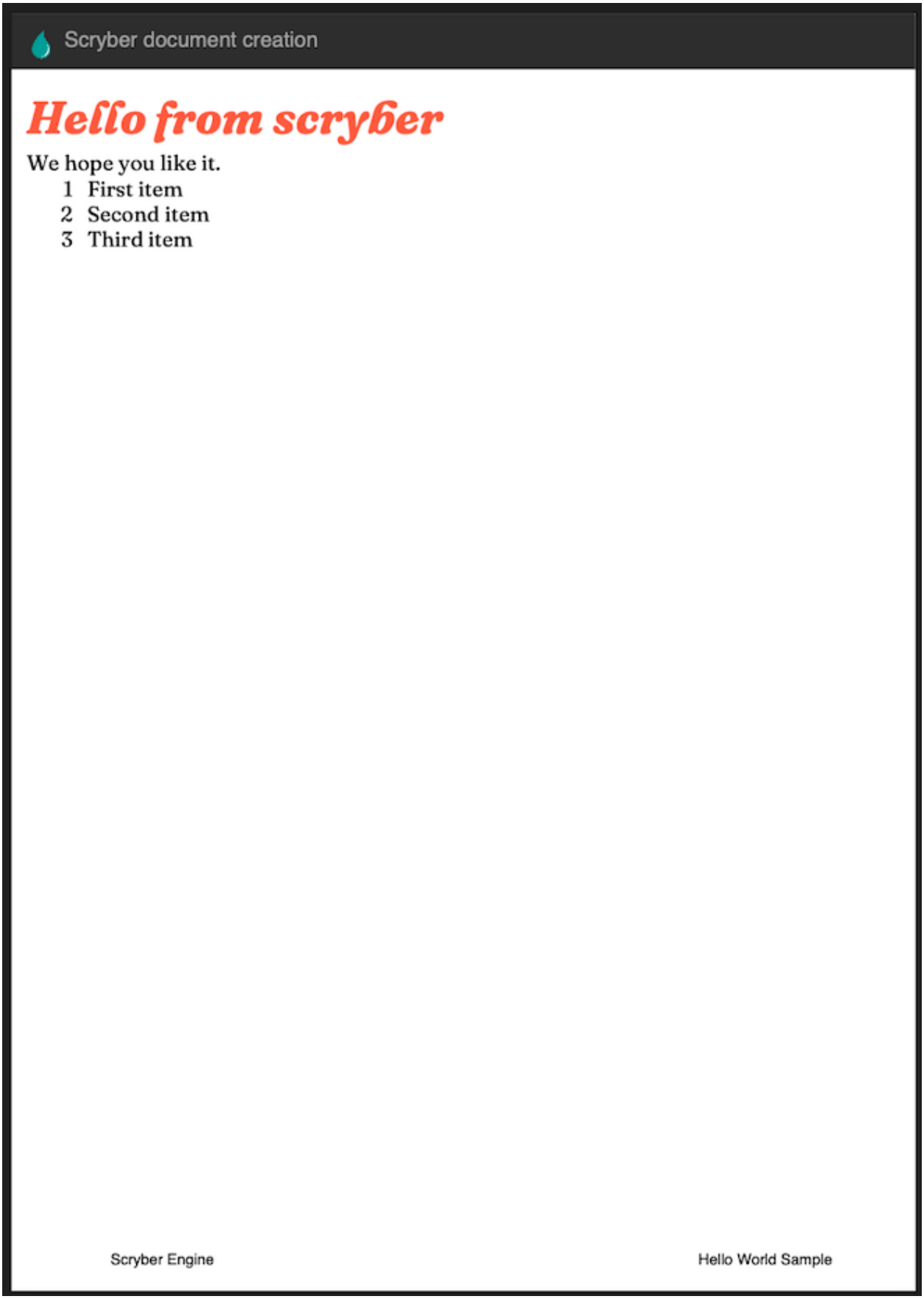
(continued from previous page)

```
</html>
```

Make some minor changes to our model.

```
using Scryber.Components;
var model = new
{
     author = "Scryber Engine",
     titlestyle = "color:#ff6347 font-family:'Fraunces'", //style data
...
```

The output from this is much more pleasing. Especially that Fruances font :-)


Scryber document creation
Hello from scryber
We hope you like it.
1 First item
2 Second item
3 Third item
Scryber Engine
Hello World Sample

10.2.10 Page Breaks and sizes

Scryber's flexible layout engine supports the use of the @media and @page css at-rules to be able to explicitly style content just for document output, and also set page sizes and orientiations.

If we add a new css styles file to our site we can alter the layout easily and know it will not affect anything else we might be using our page for.

```
/* HelloPrint.css */

@media print {

    /* We have a new page before the ordered list
     * And this new page uses the 'portrait' option */
    main ol {
        page-break-before: always;
        padding-top: 20pt;
        page: portrait;
    }

    /* Default style is A4 landscape */

    @page {
        size: A4 landscape;
    }

    /* But can use specific sizes for sections
    @page portrait {
        size: A4 portrait;
    }

}
```

And we can add a link to this in our html file. The media attribute is optional, as our css has it, but we know browsers will not load the file if it's marked for print.

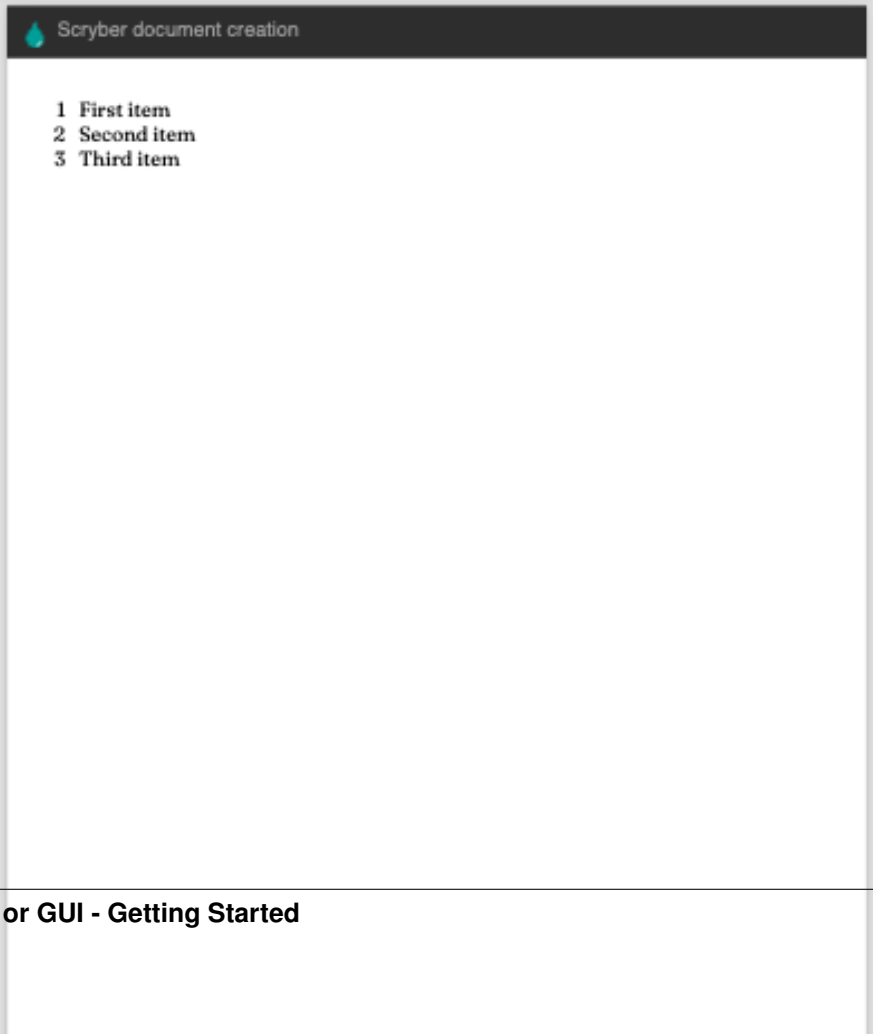
```
    </style>
    <link type='stylesheet'
        href='./CSS/HelloPrint.css'
        media='print' />
</head>
```

In our footer we can add the current page number (of total pages) and an author property directly

```
<footer>
    <table class="foot" style="width:100%">
        <tr>
            <td>{@:author}</td>

            <!-- the page tag is made up, and has a property attribute
                (open to suggestions on better syntax)  -->

            <td>Page <page /> of <page property="total" /></td>
            <td>Hello World Sample</td>
        </tr>
    </table>
</footer>
```

With this we now alter the layout to use our new pages, and everything will flow nicely.


Scryber document creation
Hello from scryber
We hope you like it.
Scryber Engine
Page 1 of 2
Hello World Sample


Scryber document creation
1 First item
2 Second item
3 Third item

10.2.11 Further reading

You can read more about the what you can do with scryber from the contents.

We have no idea what you will be able to create with scryber. It's just there to hopefully help you build amazing documents in an easy and repeatable way.

10.3 File structure

Scryber expects all content to be in valid XHTML structrure. Tags must be properly closed, and ampersands (&) must either be escaped or valid html character notations (& " etc.)

10.3.1 Example

```
<?xml version="1.0" encoding="utf-8" ?>
<?scryber append-log='true' log-level='Messages' parser-log='true' ?>
<!DOCTYPE HTML>
<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>Hello World</title>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css2?
↪family=Fraunces:ital,wght@0,400;0,700;1,400;1,700&amp;display=swap" />
    <style>
        p {
            font-family: Fraunces, 'Times New Roman', Times, serif;
            font-size: 40pt;
            border-bottom-color: aliceblue;
        }
    </style>
</head>
<body>
    <header>
        <div class="pghead">Page Header</div>
    </header>
    <div style='padding:10px; text-align:center'>
        <p>Hello World from scryber.</p>
    </div>
</body>
</html>
```

At the top of the file scryber has it's own **optional** processing instuctions to specify log levels and output. This is so you can quickly and easily check what is actually going on under the hood.

The DOCTYPE is not required, and if present will be ignored in preference to the xmlns (next).

The html tag has the xmlns attribute - this tells scryber to expect an XHTML formatted document, rather than any other document description. It is required, although you can use prefixes and any other supported namespaces (see *Drawing with SVG*)

The rest of the document follows the standard html structure, which is discussed in detail below.

10.3.2 Scryber processing instruction

The following are the supported options on the processing instruction.

- 'append-log' - Controls the tracing log output for a single document
- 'log-level' - This is an enumeration of the granularity of the logging performed on the pdf file. Values supported (from least to most) are
- 'parser-log' - Controls the logging from the xml parser.
- 'parser-culture' - specifies the global culture settings when parsing a file for interpreting dates and number formats in the content. e.g.
- 'parser-mode' - Defines how errors will be recorded if unknown or invalid attributes values are encountered.

See *Scryber Trace Log* for a detailed explanation of the tracing and logging capabilities in scryber which is **really** useful.

10.3.3 XML Namespaces

Scryber is dynamic and extensible. The xml namespaces refers directly to namespaces (and assemblies) in the library, and is reqiured. If you don't have a known namepsace, then you will get an error. There are 2 primary namespaces in use with xhtml documents.

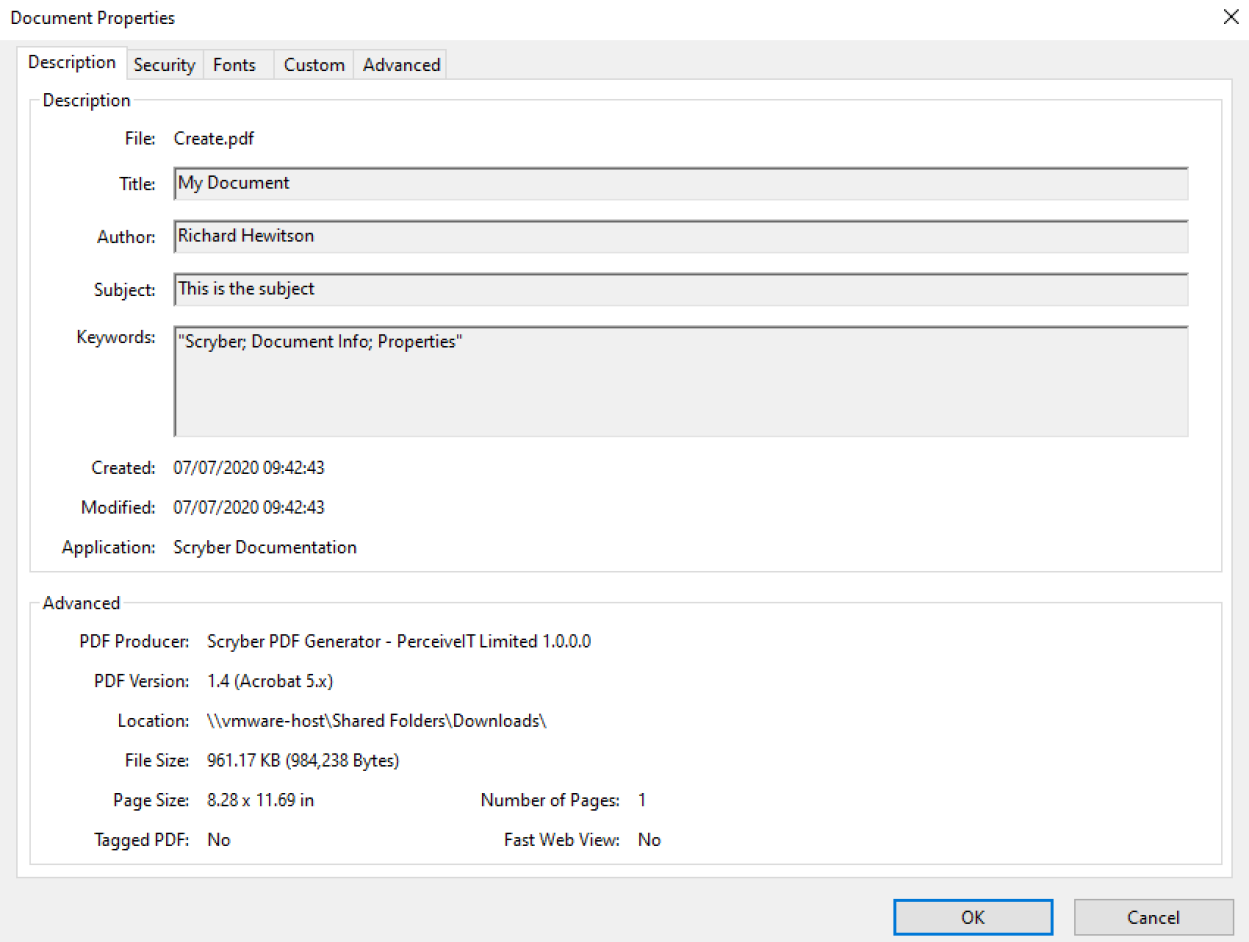
- **http://www.w3.org/1999/xhtml**
  - This is the main visual and structural components in an html file or document.
  - see *Standard document components* for a description of each of these.
- **http://www.w3.org/2000/svg**
  - These are the svg graphics components. e.g. svg, path, line, rect
  - see *Drawing with SVG* for a description of each of these.

For more information on how these are mapped, and also adding your own namespaces see *Namespaces and their Assemblies* along with scryber_configuration

10.3.4 Html header

In the html header. the following tags are supports as direct mappings to the PDF document information.

```
<head>
    <title>My Document</title>
    <base href='https://raw.githubusercontent.com/richard-scryber/scryber.core/master/
↪Scryber.Core.UnitTest/Content/HTML/' />
    <meta name='author' content='Richard Hewitson' />
    <meta name='description' content='This is the subject' />
    <meta name='keywords' content='Scryber; Document Info; Properties' />
    <meta name='generator' content='Scryber Documentation' />
    <meta name='print-restrictions' content='none' />
</head>
```



The base path will set the root path to any relative resources in the rest of the document.

The header also supports the <link> and <style> elements discussed below, although these are happily supported elsewhere too. It is only the meta and title elements that need to be in the html head. The print-restrictions applies security to the document as discussed in *Securing Documents*

10.3.5 Html link element

If a <link> is included in the html file (in the head preferably). Then it must have the 'rel' attribute of stylesheet and a 'href' to a valid css file.

Note: If the rel attribute is not set, then it is assumed to be a stylesheet, and loaded. But may not be able to be parsed.

The href can either be relative to the current file, or a full absolute url to a file.

10.3.6 Html style element

Scryber supports many of the standard html css styles. It also supports the use of @media at-rules so that css can be applied to only the document output.

More information on what css selectors are supported and the css properties see *Styles in your template*

10.3.7 The body

The body element is the visual content of the document, as 1 or more pages of content with various sizes and many features. It also suppots the header and footer elements for creating repeating headers and footers.

See *Standard document components* for a more general introduction to the visual content supported and html_tags for a full list of all the tags scyber currently supports.

10.4 Styles in your template

In scryber styles are used through out to build the document. Every component has a base style and styles (such as fill colour and font) that flow down to their inner contents.

10.4.1 Styles on elements

```
<div style='margin:20pt;padding:4pt; background-color:#FF0000; color:#FFFFFF; font-
↪family: Arial, sans-serif; font-size:20pt' >
    <span>Hello World, from scryber.</span>
</div>
```

Or if you are dynamically generating some content in the code

```
private static Component StyledComponent()
{
    var div = new Div()
    {
        BackgroundColor = new Scryber.Drawing.PDFColor(Drawing.ColorSpace.RGB, 255, 0,
↪ 0),
        Margins = new Drawing.PDFThickness(20),
        Padding = new Drawing.PDFThickness(4),
        FontFamily = "Arial",
        FontSize = 20,
        FillColor = Scryber.Drawing.PDFColors.White
    };

    div.Contents.Add(new Label()
    {
        Text = "Hello World from scryber"
    });

    return div;

}
```

10.4.2 Style Classes

Along with appling styles directly to the components, Scryber supports the use of styles declaratively and applied to the content dynamically.

```
<html xmlns='http://www.w3.org/1999/xhtml'>
    <head>
        <style>
```

(continues on next page)

(continued from previous page)

```
            div {
                font: Arial 20pt;
            }
            .mystyle {
                background-color:#FF0000;
                color:white;
                padding:20pt;
                margin:20pt;
            }
        </style>
    </head>
    <body>
        <div class="mystyle">
            <span>Hello World, from scryber.</span>
        </div>

    </body>
</html>
```

By using styles, it cleans the code and makes it easier to standardise and change later on. This can either be within the document itself, or in a separate link files (see: referencing_files)

10.4.3 Block Styles

Components such as div's, paragraphs, headings, tables, lists and list items are by default blocks. This means they will begin on a new line. Components such as spans, labels, dates and numbers are inline components. This means they will continue with the flow of content in the current line.

There are certain style attributes that will only be used on block level components. These are:

- Background Styles
- Border Styles
- Margins
- Padding
- Vertical and Horizontal alignment.

Scryber does not (currently) support inline-blocks with their associated styles, but it is in the backlog.

10.4.4 Applying Styles

Just as in css and html, styles can be applied to an element based upon (multiple) combination(s) of 3 attributes of the Style.

id class type

e.g.

```
<style>

/* This style will be applied at the document level specifying
the base level font, size and color for text. Because These
cascade down, then it will be inherited by components in the document. */
```

(continues on next page)

(continued from previous page)

```
html {
    font-family: "Gill Sans", sans-serif;
    font-size: 14pt;
    color: #333;
}

/* This style will be applied to the body tag for the first (set of) pages. */

body {
    margin: 10px;
}

/* This style will be applied to all top level headings
 specifying the font size and some spacing */

h1 {
    font-weight: bold;
    font-size: 30pt;
    margin-top: 20pt;
    padding: 5pt;
}

/* This style will be applied to all top level headings with a class of 'warning'
 and give a background colour of red on white text.  */

.warning {
    background-color: #FF0000;
    color: #FFFFFF;
}

/* This style will be applied to all components with a class of 'border'
 and give a background colour of red with white text */

.border {
    border-color: #777;
    border-width: 1pt;
    border-style: Solid;
    color: #444;
}

/* This style will be applied to all H1 Headings with a class of 'border'
 and give a border colour of red with white text. It has a higher precedence than␣
↪either h1 or .border */

h1.border {
    border-color: #550000;
    color: white;
}

/* This style will only be applied to a component with ID 'FirstHead'
 and give a font size of 48pt */

#FirstHead {
    font-size: 36pt;
    font-weight: 400;
}
```

(continues on next page)

(continued from previous page)

```
</style>
```

Note: Currently scryber does not support the concept of pseudo-classes such as :hover or :first as css e.g. div.class:first. Nor does it support !important. It may be supported in the future.

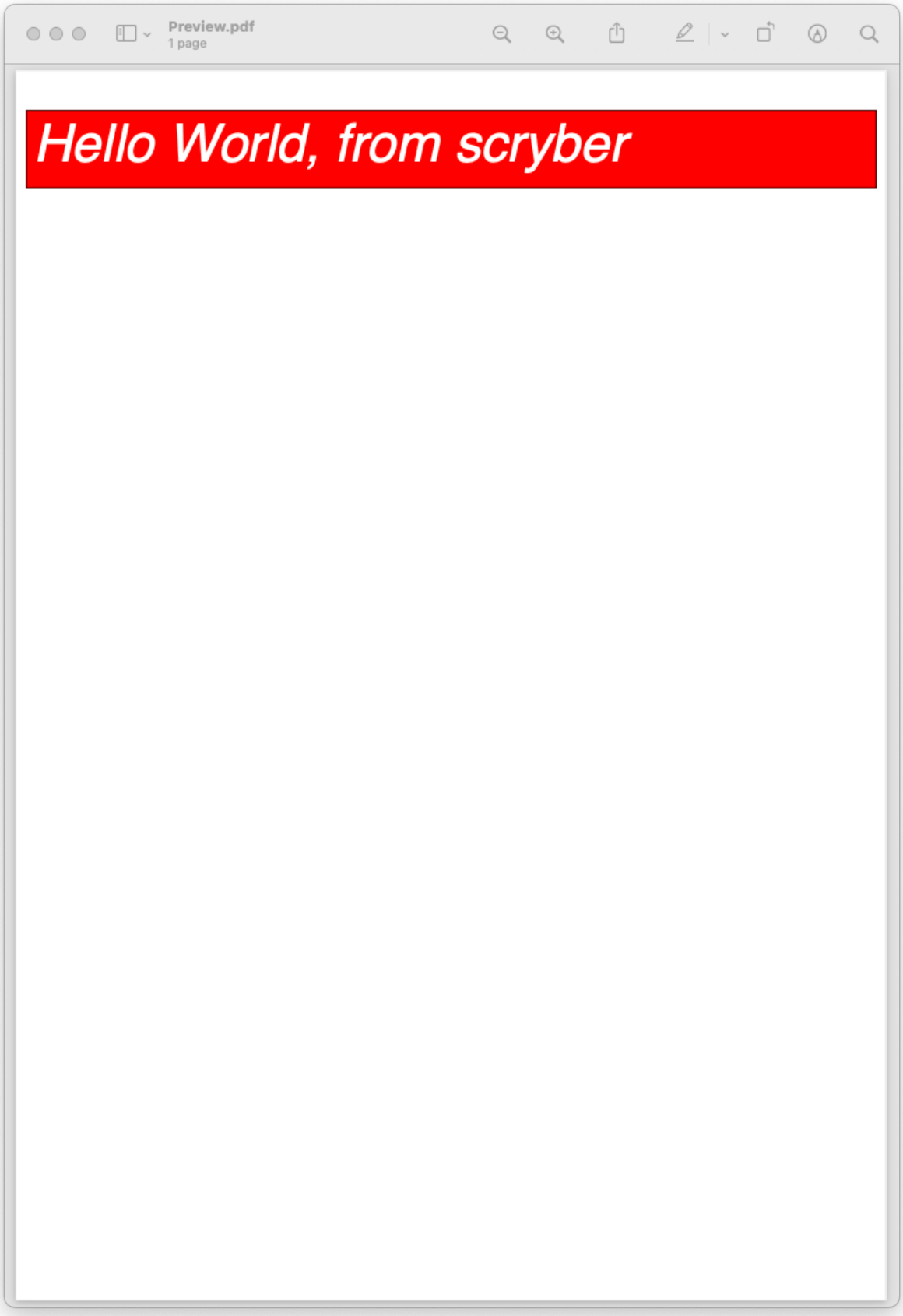
The same styles can also be applied in the code of the document styles

10.4.5 Applying Multiple Styles

Every component supports the 'class' attribute. And the value of this can be one or more class names.

```
<h1 id="FirstHead" class="warning border" style="font-italic:true" >Hello World, from␣
↪scryber</h1>
```

This will apply the h1 style, the 2 classes for the warning and border, and the h1.border applied in that precedence order and increase the size based on the ID of FirstHead. And then the inline italic style will be applied.


Preview.pdf
1 page
Hello World, from scryber

10.4.6 Late adding of styles

Even once you have parsed or built a document, the styles can still be modified or added to. Either on a component, or at a document level, as they are evaluated, allowing runtime alteration of the output.

```
//change the style sheet based on a flag check
var sheet = checkflag ? "Sheet1.css" : "Sheet2.css"

using(var doc = PDFDocument.ParseDocument("MyPath.html") as HTMLDocument)
{
    //Load the stylesheet as a referenced component
    var link = new HtmlLink(){ Href = sheet };

    //and add it to the document styles.
    doc.Head.Contents.Add(link);

    //or explicitly define a style on the document
    var defn = new StyleDefn("h1.border");
    defn.Background.Color = (PDFColor)"#FFA";
    defn.Border.Width = 2;
    defn.Border.Color = PDFColors.Red;
    defn.Border.LineStyle = LineType.Solid;

    doc.Styles.Add(defn);
}
```

10.4.7 Data binding Styles

The process of data-binding (see: document_lifecycle, and document_databinding) can apply values to styles and classes on tags.

e.g.

```
<style>

html {
    font-family: "Gill Sans", sans-serif;
    font-size: 14pt;
    color: #333;
}

body {
    margin: 10px;
}

/* this style will be applied as the bound class in the model */

.border {
    border-color: #777;
    border-width: 1pt;
    border-style: Solid;
    color: #444;
}

</style>
<body>
```

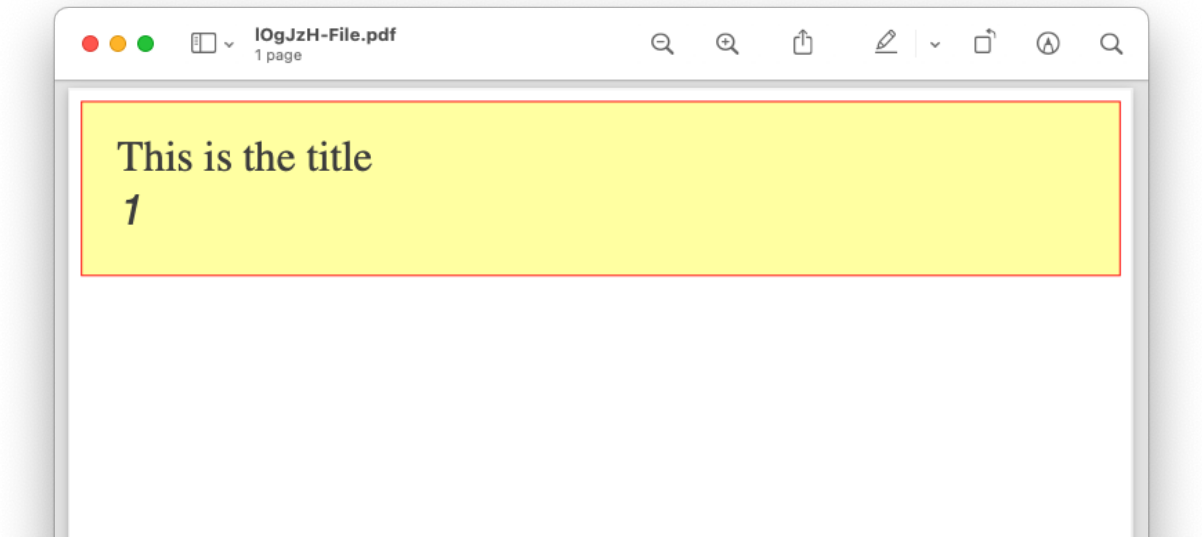
(continues on next page)

(continued from previous page)

```
    <!-- apply a theme.headclass and explicit styles -->
    <div class='{@:model.theme.headclass}' style="{@:model.theme.bg}" >

        <!-- dynamic styles for the title and number -->
        <span style="{@:model.theme.title}" >This is the title</span><br/>
        <span style="{@:model.theme.number}" >1</span>
    </div>

</body>
```

Here the theme div and spans will pick up the default theme values. Were the code can provide new style colours and fonts for output.

```
var doc = PDFDocument.ParseDocument(path);
doc.Params["model"] = new {
   theme = new {
       headclass="border",
       bg = "background-color:#FFA;padding:20pt;border:solid 1px red;",
       title = "font-family:\"Times New Roman\", Times, serif;",
       number = "font-style: italic"
    }
};

return this.PDF(doc);
```



10.4.8 Order and Precedence

Scryber tries to apply a priority, just as html to styles as they are loaded. This is based on order, depth and explicit.

div.class has a higher priority than .class

Explicit will be highest priority

And it will always fall back to the default (e.g. blue underline for anchor links).

Note: Scryber does not support !important overrides, nor does it support the use of :first-child, :hover or other pseudo classes.

Scryber has the same precedence order as html - based on the order in the document.

1. The inherited style from the parent is collected.
2. **Any styles in the document are evaluated in the order they appear.**
    1. What is the precedence of the matcher. Tag < Class < ID.
    2. What is the complexity of the match. Tag+Class < Tag+ID < Tag+Class+ID
    3. And parent selectors are evaluated to precedence Child < Parent(s) + Child
3. If a stylesheet reference is encountered, then the styles within it will be evaluated before moving on to the following styles
4. Finally the styles directly applied will be evaluated, giving the full style result.

This will then be flattened as a complete style and used in the layout and rendering of the component.

10.4.9 Supported CSS

The following CSS standard tags are supported...

- **border**
    - border-width
    - border-style
    - border-color
    - **border-top**
        - border-top-width
        - border-top-color
        - border-top-style
    - **border-left**
        - border-left-width
        - border-left-color
        - border-left-style
    - **border-right**
        - border-right-width
        - border-right-color
        - border-right-style
    - **border-bottom**
        - border-bottom-width
        - border-bottom-color
        - border-bottom-style

- color
- **background**
  - background-image
  - background-color
  - background-repeat
  - background-size
  - background-position
- **font**
  - font-style
  - font-weight - Translated to regular and bold (for the moment)
  - font-size
  - font-family
  - line-height
- **margin**
  - margin-left
  - margin-right
  - margin-top
  - margin-bottom
- **padding**
  - padding-left
  - padding-right
  - padding-top
  - padding-bottom
- opacity
- fill-opacity
- column-count
- column-gap
- column-span (for table cells)
- page-break-inside
- page-break-after
- page-break-before
- left
- top
- width
- height
- min-width

- min-height
- max-width
- max-height
- text-align
- vertical-align
- **display**
  - inline
  - block
  - none
- **overflow**
  - visible, auto
  - hidden
- **position**
  - relative
  - absolute
  - static
- text-decoration
- letter-spacing
- word-spacing
- white-space
- **list-style-type (and list-style which is treated as equivalent)**
  - bullet, disc
  - decimal
  - lower-roman
  - lower-alpha
  - upper-roman
  - upper-alpha
  - none
- **stroke**
  - stroke-opacity
  - stroke-width
- **size**
  - A4, A3, Letter, etc.
  - portrait or landscape
- **page**
  - explicit name (of an @page style)

10.4.10 at-rules supported

The following at-rules are supported

- @media - including or excluding css based on print.
- @font-face - using explicit font files and names.
- @page - specifying page sizes for sections and breaks.

10.5 Standard document components

The scryber library comes with all the standard components used in document creation, similar to HTML

10.5.1 Document level components

- **head**
  - The document description meta tags
  - Also the area to put links and styles
  - Supports the security 'restrictions' meta tags.
- **body**
  - The main pages within the document.
  - Has an optional page header and page footer, as well as content.
- **section**
  - A section can appear in the body and will by default be placed on a new page (or set of pages).
  - It supports the @page css at rule for altering the page size.

10.5.2 Structural components

- **div**
  - a block level component that will fill the width of the available parent.
- **span**
  - an inline compnent that can have any content including text.
- **table**
  - A grid of rows and cells (that can be spanned across columns).
- **ol, ul, dl**
  - Ordered lists with numbering styles.
  - Unodered lists with a bullet styles.
  - Definition lists with a label and content.
- **p(aragraph)**
  - A textual block (and other content), that has a more defined style than a div.

- **h1 to h6**
  - A textual block (and other content), that is given a pre-defined style based on it's level of 1 to 6
- **blockquote**
  - A panel with specific margins, and a left border by default.
- **pre(formatted)**
  - A container for pre-formatted text, that will not flow over new lines, or remove line breaks (by detault).
- **iframe**
  - A reference to an external file or stream that will be injected into the page(s) at runtime.
- **main**
  - A block component within the body.
  - It has an optional header and a footer.
- **page**
  - This is a non-standard component, that will be output as the

10.5.3 Textual components

Text is generally supported throughout the body of the file as you would expect.

- **b, strong**
  - An inline compenent where inner text font will be bold by default.
- **i, em**
  - An inline compenent where inner text font will be italic by default.
- **u, ins**
  - An inline compenent where inner text font will be underlined by default.
- **strike, del**
  - An inline compenent where inner text font will be strikethrough by default.
- **page**
  - A textual compenent that displays the current page number, page number of a referenced component.
  - Supports the use of page section counting and total document page count.
  - Supports the 'for' attribute to get the page number of another component.
- **time**
  - A textual component that supports the display of a date or time.
  - Can use a date-format or textual value.

10.5.4 Graphical components

- **hr**
  - A static horizontal rule on the page.
- **img**

  - An image loaded from a source, and inserted into the output document.
  - Supports the use of full, relative or dynamic url references.
  - Supports png, jpeg and tiff file formats.
  - Supports alpha channels where available in the source.
- **svg**
  - Standard drawing svg components that can be used for drawing/designs.
  - Supports the viewport and sizing options.
  - Inner content support for paths, rects, ellipses, polygons and polylines.

See *Images in documents* for images and *Drawing with SVG* for the svg componenets.

10.5.5 Data components

For a general use of the data components see document_model and document_databinding. All attributes and text can be databound

- **template**
  - Loops through one or more values in a source.
  - The data-bind attribute is used to specify the content that will be used as a source.
  - Will execute multiple times for a content within the template and the number of items binding to.
  - If it is null, then noting will be output.

10.6 Pages and Sections

All the visual content in a document sits in pages. Scryber supports the use of both a single body with content within it, and also explicit flowing pages in a section.

The use of the page-break-before is set to 'always' on a section, but can, along with page-break-after, be set and supported on any component tag

The body has an optional header and footer that will be used on every page if set.

Scryber also supports the use of the @page rule to be able to change the size and orientation of each of the pages either as a whole, or within a section or tag.

10.6.1 The body and its content

A single page has a structure of optional elements

- header - Optional, but always sited at the top of a page
- Sited between the Header and Footer is any content to be included within the page.
- footer - Optional, but always sited at the bottom of a page

If a page has a header or footer the available space for the content will be reduced.

```
<?xml version="1.0" encoding="utf-8" ?>
<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style>

    header, footer{
            padding: 10pt;
            background-color: #333;
            color: #EEE;
            border-bottom: 1px solid black;
            border-top: 1px solid black;
    }

    h1{
        padding: 20pt;
    }

    </style>
</head>
<body>
    <header>
        <h4>This is the header</h4>
    </header>
    <h1>This is the content</h1>
    <footer>
        <h4>This is the footer</h4>
    </footer>

</body>

</html>
```


This is the header
This is the content
This is the footer

Note: Any styles set on the body will be applied to the header and footer as well. e.g. padding or margins.

10.6.2 Flowing Pages

If the size of the content is more than can fit on a page it will overflow onto another page. Repeating any header or footer.

```
<?xml version="1.0" encoding="utf-8" ?>
<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style>

    header, footer {
        padding: 10pt;
        background-color: #333;
        color: #EEE;
        border-bottom: 1px solid black;
        border-top: 1px solid black;
    }

    body h1, body div {
        margin: 20pt;
    }

    body div.content {
        font-size:12pt;
        padding: 4pt;
        border: solid 1px silver;
    }

    </style>
</head>
<body>
    <header>
        <h4>This is the header</h4>
    </header>
    <h1>This is the content</h1>
    <div class='content'>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas scelerisque␣
↪porttitor urna.
    Duis pellentesque sem tempus magna faucibus, quis lobortis magna aliquam. Nullam␣
↪eu risus
    facilisis sapien fermentum condimentum. Pellentesque ut placerat diam, sed␣
↪suscipit nibh.
    Integer dictum dolor vel finibus imperdiet. Orci varius natoque penatibus et␣
↪magnis dis
    parturient montes, nascetur ridiculus mus. Integer congue turpis at varius␣
↪porttitor.
    <!-- Truncated for brevity -->
    nec faucibus ipsum bibendum sed. Nunc tristique risus eu quam porttitor blandit.
    In erat mauris, imperdiet a venenatis eu, tempus a nunc.
    <br/>
    Nullam et erat vel nisl suscipit volutpat id vitae massa. Nunc volutpat feugiat␣
↪iaculis.
```

(continues on next page)

(continued from previous page)

```
    Mauris sit amet eleifend augue. Nulla imperdiet eu mauris nec consequat. Donec a␣
↪urna blandit,
    porttitor libero vel, rutrum diam. Fusce scelerisque diam eu rutrum vestibulum.
    Vivamus a quam in nisi euismod laoreet. Morbi mauris augue, lobortis id volutpat␣
↪in,
    venenatis ut ex. Donec euismod risus eros, dapibus tincidunt dolor varius id.
    </div>
    <footer>
        <h4>This is the footer</h4>
    </footer>

</body>

</html>
```

Here we can see that the content flows naturally onto the next page, including the padding and borders. And the header and footer are shown on the second page.


This is the header
This is the content
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas scelerisque porttitor urna. Duis pellentesque sem tempus magna faucibus, quis lobortis magna aliquam. Nullam eu risus facilisis sapien fermentum condimentum. Pellentesque ut placerat diam, sed suscipit nibh. Integer dictum dolor vel finibus imperdiet. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer congue turpis at varius porttitor. Sed mattis mi et nunc porttitor fermentum. Sed porta eu urna et varius. Suspendisse sit amet nibh quis massa sodales egestas tempor sit amet tortor. Sed sed hendrerit lacus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Praesent mollis tempor enim.
Duis at porttitor tortor, vitae consequat ligula. Pellentesque ut nulla cursus, dictum massa ac, aliquet ipsum. Nullam rutrum, odio id venenatis ultrices, odio ipsum placerat elit, sed fermentum purus est nec nisl. Duis dictum ex auctor, varius mauris at, scelerisque ligula. Donec congue scelerisque sagittis. Suspendisse consequat, orci ut vulputate consectetur, nunc justo tincidunt arcu, et suscipit sem augue ac sem. Suspendisse suscipit, nisl ac elementum ornare, dui mi placerat nibh, quis faucibus sapien felis ac arcu. Curabitur facilisis metus non vestibulum laoreet. Praesent in urna vel mi ultrices pharetra sed in ipsum. Pellentesque placerat dui et iaculis dapibus. Mauris laoreet elit vitae diam molestie, at pulvinar nunc interdum.
Nullam et erat vel nisl suscipit volutpat id vitae massa. Nunc volutpat feugiat iaculis. Mauris sit amet eleifend augue. Nulla imperdiet eu mauris nec consequat. Donec a urna blandit, porttitor libero vel, rutrum diam. Fusce scelerisque diam eu rutrum vestibulum. Vivamus a quam in nisi euismod laoreet. Morbi mauris augue, lobortis id volutpat in, venenatis ut ex. Donec euismod risus eros, dapibus tincidunt dolor varius id. Quisque congue odio eget massa tincidunt iaculis id vel elit. Vivamus sodales, dolor nec tempor sodales, lectus urna viverra tortor, a lacinia magna diam et est.
Aliquam erat volutpat. Nam finibus rhoncus velit, in dignissim metus finibus et. In dignissim nisi nec dapibus volutpat. Proin ultrices nibh a gravida euismod. Donec commodo erat at ipsum commodo, a ultricies odio vulputate. Nunc efficitur dui non faucibus pretium. Maecenas sollicitudin sem sagittis dignissim volutpat. Integer efficitur euismod lectus at varius. Vestibulum euismod massa mauris. Mauris laoreet urna est, et tristique velit lobortis eu.
Phasellus luctus dapibus nisi, et pulvinar neque ultrices vitae. Pellentesque quis purus felis. Aliquam feugiat efficitur sem quis placerat. Quisque viverra magna vitae elit eleifend, a porttitor enim vulputate. Quisque elit metus, aliquam eget purus at, blandit gravida diam. Vivamus placerat blandit libero, in pellentesque augue accumsan eu. Vivamus at ante ut elit dignissim fermentum vel vitae leo.
Vestibulum et ligula ligula. Nullam placerat orci arcu, nec faucibus ipsum bibendum sed. Nunc tristique risus eu quam porttitor blandit. In erat mauris, imperdiet a venenatis eu, tempus a nunc.
Aliquam erat volutpat. Nam finibus rhoncus velit, in dignissim metus finibus et. In dignissim nisi nec dapibus volutpat. Proin ultrices nibh a gravida euismod. Donec commodo erat at ipsum commodo, a ultricies odio vulputate. Nunc efficitur dui non faucibus pretium. Maecenas sollicitudin sem sagittis dignissim volutpat. Integer efficitur euismod lectus at varius. Vestibulum euismod massa mauris. Mauris laoreet urna est, et tristique velit lobortis eu.
Phasellus luctus dapibus nisi, et pulvinar neque ultrices vitae. Pellentesque quis purus felis. Aliquam feugiat efficitur sem quis placerat. Quisque viverra magna vitae elit eleifend, a porttitor enim vulputate. Quisque elit metus, aliquam eget purus at, blandit gravida diam. Vivamus placerat blandit libero, in
This is the footer
This is the header
pellentesque augue accumsan eu. Vivamus at ante ut elit dignissim fermentum vel vitae leo.
Vestibulum et ligula ligula. Nullam placerat orci arcu, nec faucibus ipsum bibendum sed. Nunc tristique risus eu quam porttitor blandit. In erat mauris, imperdiet a venenatis eu, tempus a nunc.
Nullam et erat vel nisl suscipit volutpat id vitae massa. Nunc volutpat feugiat iaculis. Mauris sit amet eleifend augue. Nulla imperdiet eu mauris nec consequat. Donec a urna blandit, porttitor libero vel, rutrum diam. Fusce scelerisque diam eu rutrum vestibulum. Vivamus a quam in nisi euismod laoreet. Morbi mauris augue, lobortis id volutpat in, venenatis ut ex. Donec euismod risus eros, dapibus tincidunt dolor varius id.

10.6.3 Page breaks

When using a <section> it will, by default, force a break in the pages using the before the component, so that it flows nicely onto a new page and begins the new content from there. (the default style is page-break-before:always)

This behaviour can can be stopped by applying the css attribute for 'page-break-before:avoid' value, and a page break can also be applied to any element using the style 'page-break-before:always' (or 'page-break-after:always').

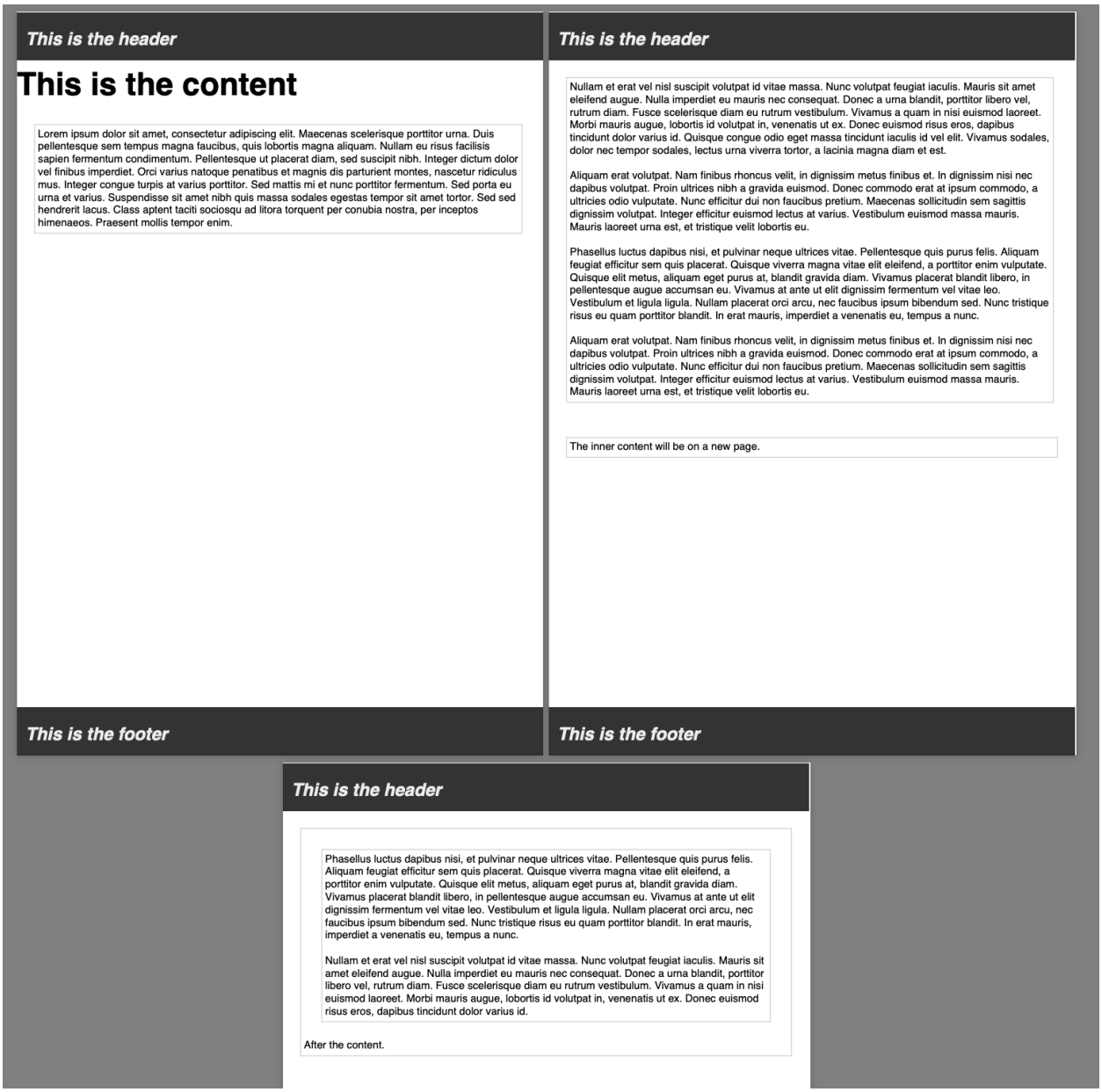
Margins, padding, boarder and depth should be preserved during the page break, and the engine will try and layout the content appropriately for breaks inside nested elements.

```
<?xml version="1.0" encoding="utf-8" ?>
<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">

        header, footer {
        padding: 10pt;
        background-color: #333;
        color: #EEE;
        border-bottom: 1px solid black;
        border-top: 1px solid black;
        }

        body .content {
        margin: 20pt;
        font-size:12pt;
        padding: 4pt;
        border: solid 1px silver;
        }

    </style>
</head>
<body>
    <header>
        <h4>This is the header</h4>
    </header>
    <h1>This is the content</h1>

    <!-- section that does not force a new page (so that it stays on the first page --
↪>
    <section class='content' style="page-break-before:avoid">
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas scelerisque␣
↪porttitor urna.
        <!-- Truncated for brevity -->
        Class aptent taciti sociosqu ad litora torquent per conubia nostra, per␣
↪inceptos himenaeos.
        Praesent mollis tempor enim.<br />

    </section>

    <!-- This will be default appear on a new page -->
    <section class='content'>
        Nullam et erat vel nisl suscipit volutpat id vitae massa. Nunc volutpat␣
↪feugiat iaculis.
        Mauris sit amet eleifend augue. Nulla imperdiet eu mauris nec consequat.␣
↪Donec a urna blandit,
        <!-- Truncated for brevity -->
```

(continues on next page)

(continued from previous page)

```
        sagittis dignissim volutpat. Integer efficitur euismod lectus at varius.␣
↪Vestibulum euismod massa mauris.
        Mauris laoreet urna est, et tristique velit lobortis eu.
    </section>

    <!-- Any tag can force a new page within the document flow, and it does not have␣
↪to be at the
        root level. Borders and spacing will be preserved as much as possible -->
    <div class="content">
        The inner content will be on a new page.
        <div class='content' style="page-break-before:always;">
            Phasellus luctus dapibus nisi, et pulvinar neque ultrices vitae.␣
↪Pellentesque quis purus felis.
            <!-- Truncated for brevity -->
            venenatis ut ex. Donec euismod risus eros, dapibus tincidunt dolor varius␣
↪id.
        </div>
        After the content.
    </div>
    <footer>
        <h4>This is the footer</h4>
    </footer>

</body>

</html>
```



10.6.4 Page size and orientation

When outputting a page the default paper size is ISO A4 Portrait (210mm x 29.7mm), however Scryber supports setting the paper size either on the section or via styles to the standard ISO or Imperial page sizes, in landscape or portrait.

- **ISO 216 Standard Paper sizes**
  - A0 to A9
  - B0 to B9
  - C0 to C9
- **Imperial Paper Sizes**

  - Quarto, Foolscap, Executive, GovermentLetter, Letter, Legal, Tabloid, Post, Crown, LargePost, Demy, Medium, Royal, Elephant, DoubleDemy, QuadDemy, Statement,

The body or a section can only be 1 size of paper, but different sections (or page breaks) can be different pages and can have different sizes.

An @page { ... } rule will apply to all pages in the document.

To specify an explicit named page size use the name after the @page rule, and then identify the rule with the page css declaration either on the tag style or in css. The same priories will be applied if multiple page values are matched.

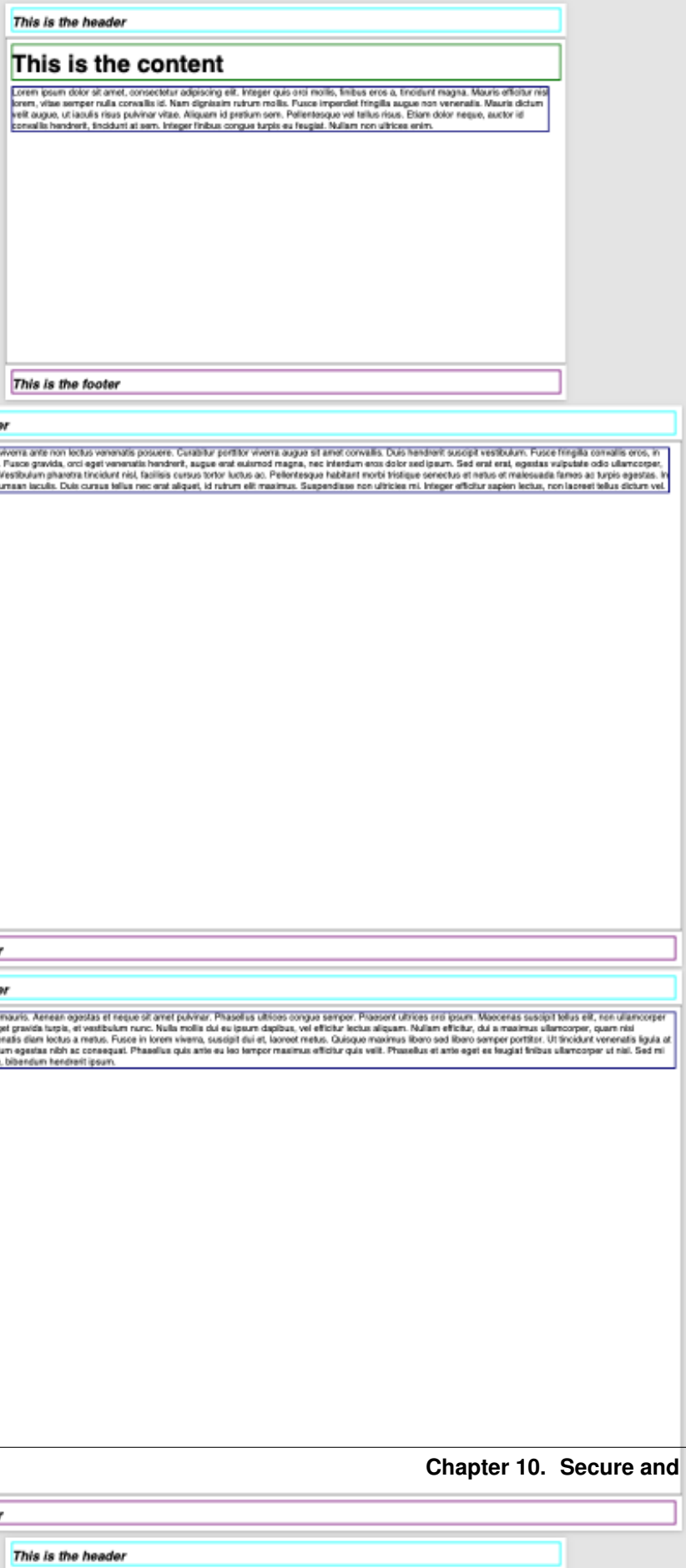
To revert back to the default size use a value of auto or initial.

```
<?xml version="1.0" encoding="utf-8" ?>
<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style>
        /* This changes the default page size to A4 landscape */
        @page {
            size: A4 landscape;
        }

        /* this is an explicit style of page size as A3 */
        @page large {
            size: A3 landscape;
        }

    </style>
</head>
<body style='border:solid 1px gray;padding:5pt;'>
    <header>
        <h4 style='margin: 5pt; border-width: 1pt; border-color:aqua'>This is the␣
↪header</h4>
    </header>
    <h1 style='margin:5pt; border-width: 1pt; border-color: green;'>This is the␣
↪content</h1>
    <!-- Set a section to not break on the first page -->
    <section style="page-break-before: avoid; margin:5pt; font-size: 14pt; border-
↪width: 1pt; border-color: navy;">
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer quis orci␣
↪mollis, finibus eros a,
        .....
    </section>
    <!-- By default this will start on a new page with A3 size -->
    <section style="page:large; margin:5pt; font-size: 14pt; border-width: 1pt;␣
↪border-color: navy;">
        In ac diam sapien. Morbi viverra ante non lectus venenatis posuere. Curabitur␣
↪porttitor viverra augue
        sit amet convallis. Duis hendrerit suscipit vestibulum. Fusce fringilla␣
↪convallis eros, in vehicula
        .....
        Integer efficitur sapien lectus, non laoreet tellus dictum vel.<br />
        <!-- Introducing an inner page break that follow the same A3 size -->
        <div style="page-break-before:always">
            Maecenas vitae vehicula mauris. Aenean egestas et neque sit amet pulvinar.
            Phasellus ultrices congue semper. Praesent ultrices orci ipsum. Maecenas␣
↪suscipit tellus elit,
            .....
        </div>
```

(continues on next page)

(continued from previous page)

```
    </section>
    <!-- Outside of the large page section use page: auto to revert to the default␣
↪size -->
    <div style="page-break-before:always; page: auto;">
        Maecenas vitae vehicula mauris. Aenean egestas et neque sit amet pulvinar.
        Phasellus ultrices congue semper. Praesent ultrices orci ipsum. Maecenas␣
↪suscipit tellus elit,
        .....
    </div>
    <footer>
        <h4 style="margin:5pt; border-width: 1pt; border-color: purple;">This is the␣
↪footer</h4>
    </footer>
</body>

</html>
```


This is the header
This is the content
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer quis orci mollis, finibus eros a, tincidunt magna. Mauris efficitur nisl lorem, vitae semper nulla convallis id. Nam dignissim rutrum mollis. Fusce imperdiet fringilla augue non venenatis. Mauris dictum velit augue, ut iaculis risus pulvinar vitae. Aliquam id pretium sem. Pellentesque vel tellus risus. Etiam dolor neque, auctor id convallis hendrerit, tincidunt at sem. Integer finibus congue turpis eu feugiat. Nullam non ultrices enim.
This is the footer
This is the header
In ac diam sapien. Morbi viverra ante non lectus venenatis posuere. Curabitur porttitor viverra augue sit amet convallis. Duis hendrerit suscipit vestibulum. Fusce fringilla convallis eros, in vehicula nibh tempor sed. Fusce gravida, orci eget venenatis hendrerit, augue erat euismod magna, nec interdum eros dolor sed ipsum. Sed erat erat, egestas vulputate odio ullamcorper, vulputate dapibus libero. Vestibulum pharetra tincidunt nisl, facilisis cursus tortor luctus ac. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. In auctor quam vel nibh accumsan iaculis. Duis cursus tellus nec erat aliquet, id rutrum elit maximus. Suspendisse non ultricies mi. Integer efficitur sapien lectus, non laoreet tellus dictum vel.
This is the footer
This is the header
Maecenas vitae vehicula mauris. Aenean egestas et neque sit amet pulvinar. Phasellus ultrices congue semper. Praesent ultrices orci ipsum. Maecenas suscipit tellus elit, non ullamcorper nulla blandit sed. Nulla eget gravida turpis, et vestibulum nunc. Nulla mollis dui eu ipsum dapibus, vel efficitur lectus aliquam. Nullam efficitur, dui a maximus ullamcorper, quam nisi imperdiet sapien, ac venenatis diam lectus a metus. Fusce in lorem viverra, suscipit dui et, laoreet metus. Quisque maximus libero sed libero semper porttitor. Ut tincidunt venenatis ligula at viverra. Phasellus bibendum egestas nibh ac consequat. Phasellus quis ante eu leo tempor maximus efficitur quis velit. Phasellus et ante eget ex feugiat finibus ullamcorper ut nisl. Sed mi nunc, blandit ut sem vitae, bibendum hendrerit ipsum.
This is the footer
This is the header

10.6.5 Stopping overflow

If overflowing onto a new page is not required or wanted then the page-break-inside='avoid' will block any overflow or new pages.

A section can be a single page, and never overflow.

10.7 Pages Numbers

Putting numbers in pages is often a requirement, but honestly we have never liked the CSS approach.

At scryber we have taken a slightly more declarative approach with the 'page' tag. Browsers do not understand this tag, and will ignore it. The scryber engine will understand and output the current page number

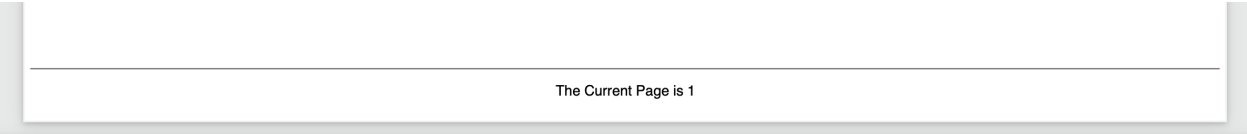
10.7.1 Current Page Numbers

```
<footer>
    <p class='print-only foot'> The Current Page is <page /></p>
</footer>
```

We can add our explicit styles to this, so the content of the footer will not be shown by browsers But will display nicely on the document.

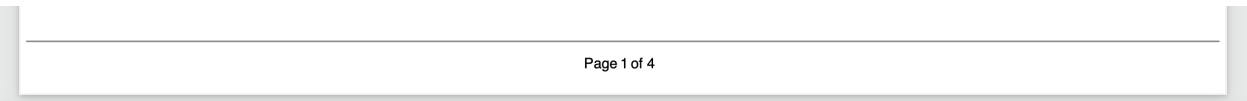
```
.print-only{ display:none;}

@media print{

    .print-only{ display: block; }

    .foot{
        border-top: solid 1pt gray;
        padding: 10pt;
        text-align:center;
        font: 10pt sans-serif;
        margin: 5pt;
    }
}
```

And our page footer will display as expected.



The page tag also supports the property attribute for displying the 'total' number of pages, and also the current 'section' or 'sectiontotal' page count.

```
<footer>
    <p class='print-only foot'> Page <page /> of <page property='total' /></p>
</footer>
```



10.7.2 The page for

Conversly to the current page number, it is also possible to get the page number of another element. By using the 'for' attribute.

The example below is a table of contents with links to sections based on their ID and a line leading to the page numbers on the right cell.

```
<table class="toc" style="margin:20pt; width:100%;">
    <thead>
        <tr>
            <td colspan="2">Table of contents</td>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td><a href="#section1">Section 1</a><hr class="tab-spacer" /></td>
            <td class="pg-num"><page for="#section1" /></td>
        </tr>
        <tr>
            <td><a href="#section2">Section 2</a><hr class="tab-spacer" /></td>
            <td class="pg-num"><page for="#section2" /></td>
        </tr>
        <tr>
            <td><a href="#section3">Section 1</a><hr class="tab-spacer" /></td>
            <td class="pg-num"><page for="#section3" /></td>
        </tr>
    </tbody>
</table>
```

There is a bit of css fun going on to achieve this..

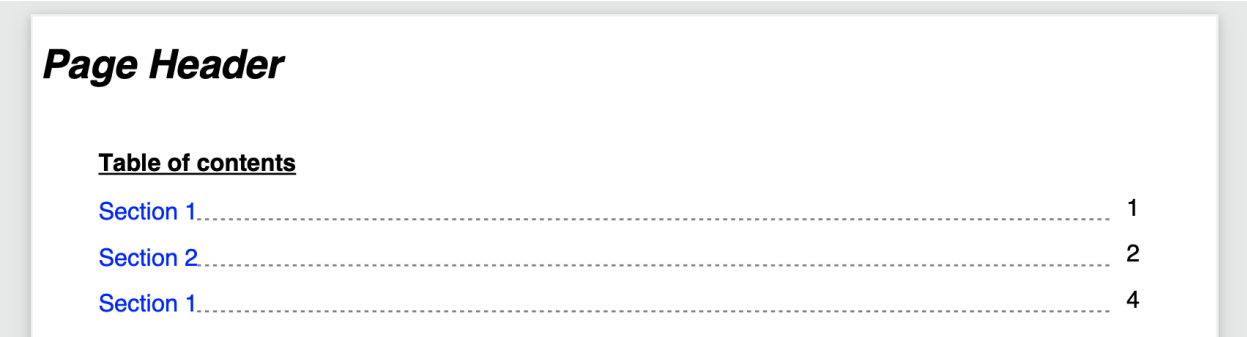
```
table.toc{
    font-size:12pt;
    margin-left:30pt;
}

table.toc thead{
    font-weight:bold;
    text-decoration:underline;
}

table.toc a{
    text-decoration:none;
}

table.toc hr{
    display:inline;
    margin-top:12pt;
    stroke: gray;
    stroke-dasharray: 2;
}

table.toc td{
    border:none;
}

table.toc td.pg-num {
```

(continues on next page)

(continued from previous page)

```
    width:30pt;
}
```

But the output is quite pleasing. And you could use databinding to achieve this.



10.8 Column layout

Scryber supports flowing content layout. No matter the font, content type or structure. (see *Pages and Sections*) It also supports the same with columns, at the page and container level.

10.8.1 Specifying columns on Pages

All block level components (see *Positioning your content*) support the use of columns.

There is by default on a block a single column, but by specifying a style:column-count (either on the block, or in the style definition) then the layout will split the block into that number of regions within it.

Content within the column will flow down as far as it is able (either the bottom or the page, or the maximum height of the container) and then move to the top of the next column.

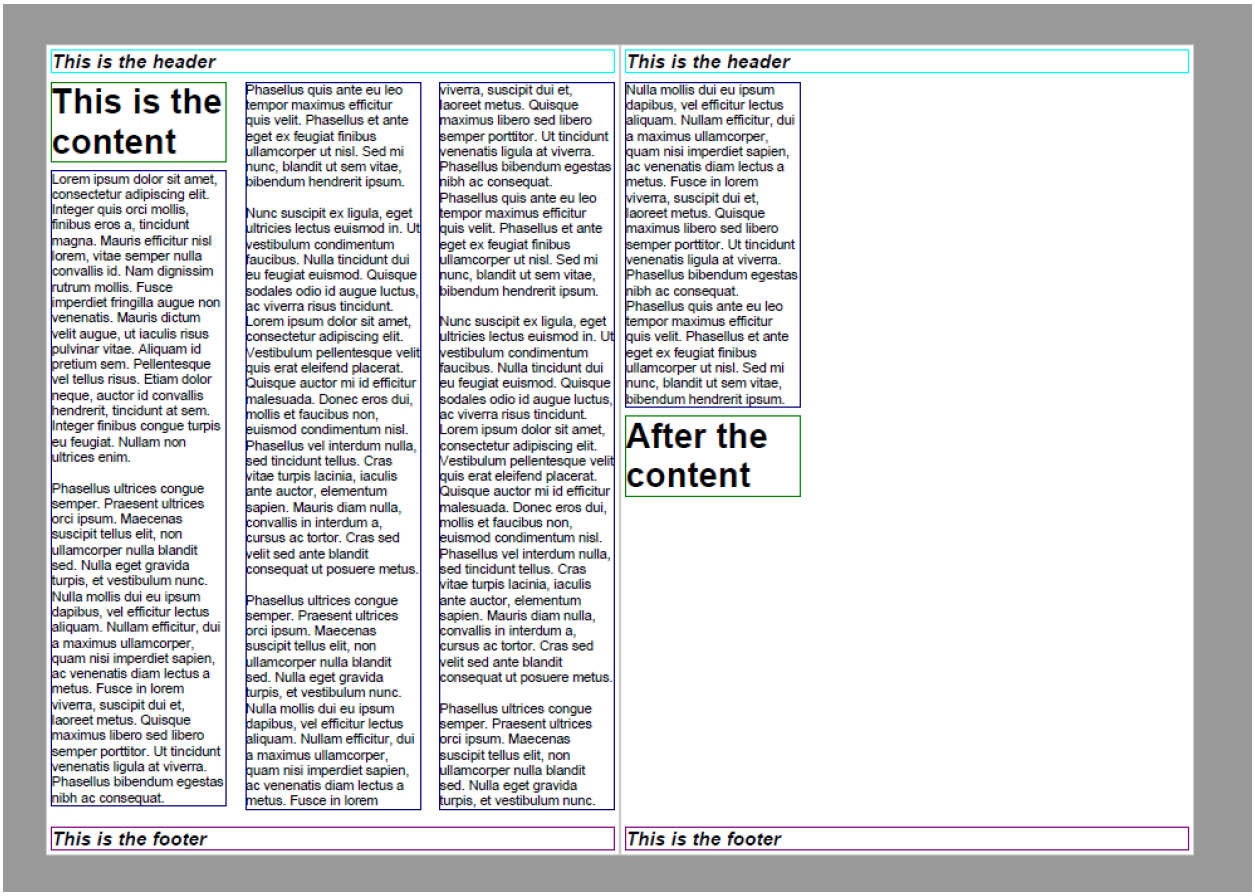
Below we specify the section to have 3 columns, they will be of equal size and the content within it will flow onto each column and then ulimately the next page.

```
<html xmlns='http://www.w3.org/1999/xhtml'>
<!-- set the coumn count on the body -->
<body style="column-count:3;">
    <header>
        <h4 style='margin: 5pt; border-bottom-width: 1pt; border-bottom-color:aqua'>
↪This is the header</h4>
    </header>

    <h1 style='margin:5pt;border-width:1pt; border-color:green'>This is the content</
↪h1>

    <div style='margin:5pt; font-size:14pt; border-width: 1pt; border-color: navy'>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer quis orci␣
↪mollis, finibus eros a,
        tincidunt magna. Mauris efficitur nisl lorem, vitae semper nulla convallis id.
↪ Nam dignissim rutrum
        mollis. Fusce imperdiet fringilla augue non venenatis. Mauris dictum velit␣
↪augue, ut iaculis risus
        pulvinar vitae. Aliquam id pretium sem. Pellentesque vel tellus risus. Etiam␣
↪dolor neque, auctor id
```

(continues on next page)

(continued from previous page)

```
        convallis hendrerit, tincidunt at sem. Integer finibus congue turpis eu␣
↪feugiat. Nullam non ultrices enim.<br />
        <!-- Truncated for brevity
        .
        . -->
        Maecenas vitae vehicula mauris. Aenean egestas et neque sit amet pulvinar.
        Phasellus ultrices congue semper. Praesent ultrices orci ipsum. Maecenas␣
↪suscipit tellus elit,
        non ullamcorper nulla blandit sed. Nulla eget gravida turpis, et vestibulum␣
↪nunc. Nulla mollis
        dui eu ipsum dapibus, vel efficitur lectus aliquam. Nullam efficitur, dui a␣
↪maximus ullamcorper,
        quam nisi imperdiet sapien, ac venenatis diam lectus a metus. Fusce in lorem␣
↪viverra, suscipit
        dui et, laoreet metus. Quisque maximus libero sed libero semper porttitor. Ut␣
↪tincidunt venenatis
        ligula at viverra. Phasellus bibendum egestas nibh ac consequat. Phasellus␣
↪quis ante eu leo tempor
        maximus efficitur quis velit. Phasellus et ante eget ex feugiat finibus␣
↪ullamcorper ut nisl. Sed mi
        nunc, blandit ut sem vitae, bibendum hendrerit ipsum.<br />
    </div>
    <h1 style="margin:5pt; border-width:1pt; border-color: green">After the content</
↪h1>

    <footer>
        <h4 style="margin: 5pt; border-width: 1pt; border-color: purple">This is the␣
↪footer</h4>
    </footer>
</body>

</html>
```



Here the page is set to 3 columns across the layout page. The headers are independent of the column setting, but the inner content flows down the first column, and then moves to the next when no more can be fitted, and so on until it reaches the end of the layout page.

As we can overflow, a new layout page is added, and the content latout continues until the end.

Note: The borders and any background will be based around the container and show on each column.

10.8.2 Columns on containers

As we can see above, the headings were also part of the column layout on the page.

Scryber supports the use of columns on containers too. So our layout can be improved if we remove the columns from the page, and set them on the *div* itself.

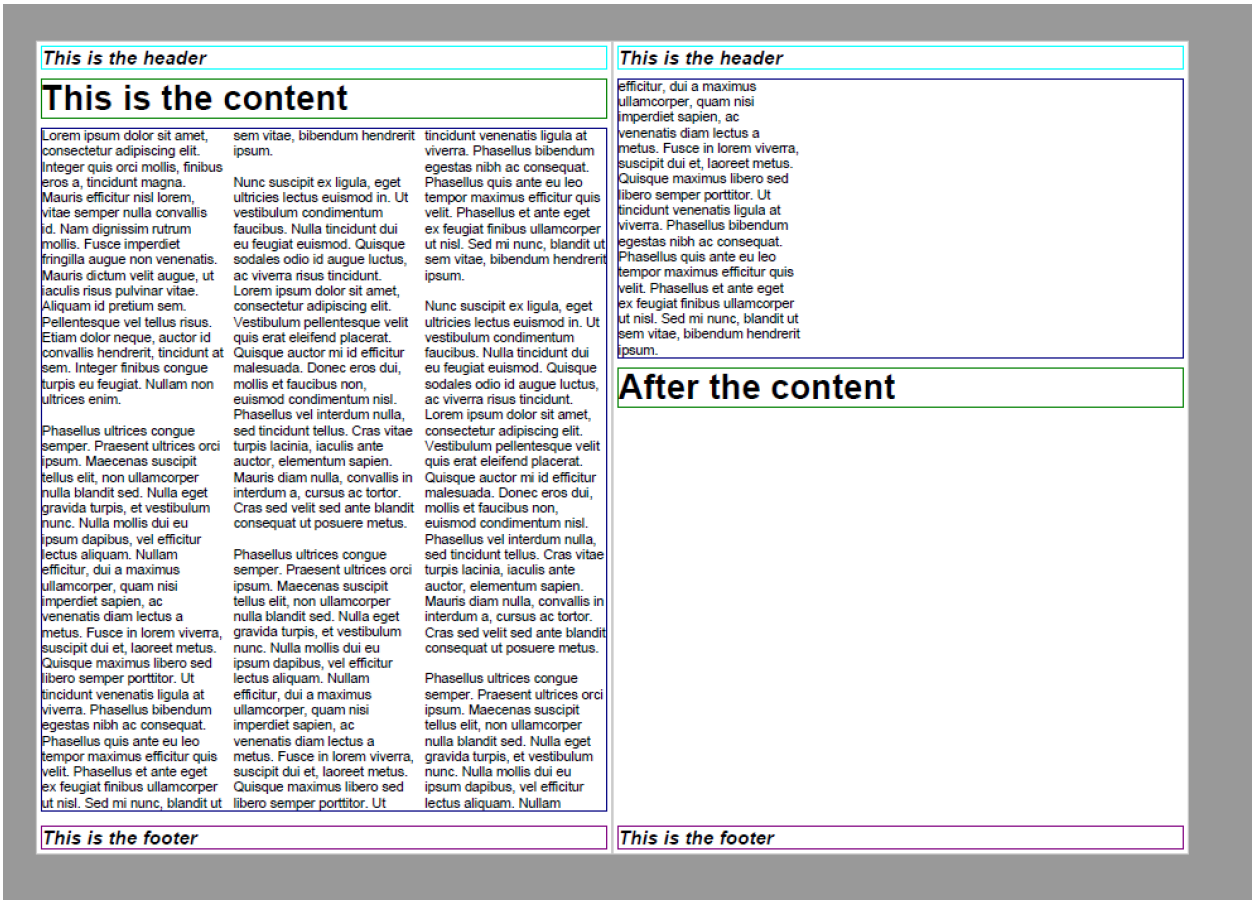
This will allow the headers to be full width, with the content flowing within the columns of the container.

```
<html xmlns='http://www.w3.org/1999/xhtml'>
<!-- remove the coumn count on the body -->
<body>
    <header>
        <h4 style='margin: 5pt; border-bottom-width: 1pt; border-bottom-color:aqua'>
↪This is the header</h4>
    </header>
    <h1 style='margin:5pt;border-width:1pt; border-color:green'>This is the content</
↪h1>
```

(continues on next page)

(continued from previous page)

```
    <!-- Set the div to have 3 columns
        rather than the page -->

    <div style='column-count:3; margin:5pt; font-size:14pt; border-width: 1pt; border-
↪color: navy'>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer quis orci␣
↪mollis, finibus eros a,
        tincidunt magna. Mauris efficitur nisl lorem, vitae semper nulla convallis id.
↪ Nam dignissim rutrum
        mollis. Fusce imperdiet fringilla augue non venenatis. Mauris dictum velit␣
↪augue, ut iaculis risus
        pulvinar vitae. Aliquam id pretium sem. Pellentesque vel tellus risus. Etiam␣
↪dolor neque, auctor id
        convallis hendrerit, tincidunt at sem. Integer finibus congue turpis eu␣
↪feugiat. Nullam non ultrices enim.<br />
        <!-- Truncated for brevity
        .
        . -->
        Maecenas vitae vehicula mauris. Aenean egestas et neque sit amet pulvinar.
        Phasellus ultrices congue semper. Praesent ultrices orci ipsum. Maecenas␣
↪suscipit tellus elit,
        non ullamcorper nulla blandit sed. Nulla eget gravida turpis, et vestibulum␣
↪nunc. Nulla mollis
        dui eu ipsum dapibus, vel efficitur lectus aliquam. Nullam efficitur, dui a␣
↪maximus ullamcorper,
        quam nisi imperdiet sapien, ac venenatis diam lectus a metus. Fusce in lorem␣
↪viverra, suscipit
        dui et, laoreet metus. Quisque maximus libero sed libero semper porttitor. Ut␣
↪tincidunt venenatis
        ligula at viverra. Phasellus bibendum egestas nibh ac consequat. Phasellus␣
↪quis ante eu leo tempor
        maximus efficitur quis velit. Phasellus et ante eget ex feugiat finibus␣
↪ullamcorper ut nisl. Sed mi
        nunc, blandit ut sem vitae, bibendum hendrerit ipsum.<br />
    </div>
    <!-- after the columns we go full width again -->
    <h1 style="margin:5pt; border-width:1pt; border-color: green">After the content</
↪h1>

    <footer>
        <h4 style="margin: 5pt; border-width: 1pt; border-color: purple">This is the␣
↪footer</h4>
    </footer>
</body>
```



10.8.3 Column-width and column-gap

Rather than specifying the number of columns, scryber also supports the standard html column-width option.

This makes the width value the predominant driver, and will layout the maximum number of columns that are at least this width within the available space, so that it is full width.

e.g. if you have a container that is 300pts wide and a column-width of 80pt, then there will be 3 columns of about 92pts wide (assuming the alley / gap is the default 10pt). Increasing the column-width to 120pt, and the number of columns will reduce to 2 of around 145pts.

If our page size or orientation changes then the number of columns fitted changes.

Column-gaps are the margins, or alleys, between each column. The default is 10pt, but it can be specified as a single unit value, e.g. 20pt or 5mm (see *Units and measures* for more on scryber measurements).

The break-before and break-after values of 'always' will cause the flow of the content to move to the next available column. If all the columns are used, then a new page will be created and the content continue to flow from there.

Finally, the break-inside value of 'avoid' will try to keep all the inner content in one block. If it does not fit within the current column, then it will move to a new column (or column on a new page) and continue from there.

```
<?xml version="1.0" encoding="utf-8" ?>
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">

   <html xmlns='http://www.w3.org/1999/xhtml'>
```

(continues on next page)

(continued from previous page)

```
   <head>
       <style type="text/css" >

           @page {
           size: A4 Landscape;
           }

       </style>
   </head>
   <body>
       <header>
           <h4 style='margin: 5pt; border-bottom-width: 1pt; border-bottom-color:aqua
↪'>This is the header</h4>
       </header>
       <h1 style='margin:5pt;border-width:1pt; border-color:green'>This is the content
↪</h1>
       <!-- Set a section to not break on the first page -->
       <div style='column-width: 150pt; column-gap:60pt; margin:5pt; font-size: 11pt;␣
↪border-width: 1pt; border-color: navy'>
           Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer quis orci␣
↪mollis, finibus eros a,
           tincidunt magna. Mauris efficitur nisl lorem, vitae semper nulla convallis␣
↪id. Nam dignissim rutrum
           ......
           <div style="break-before:always;" >
               In ac diam sapien. Morbi viverra ante non lectus venenatis posuere.␣
↪Curabitur porttitor viverra augue sit amet
               convallis. Duis hendrerit suscipit vestibulum. Fusce fringilla␣
↪convallis eros, in vehicula nibh tempor sed.
               ........
           </div>
           <div style="break-inside:avoid">
               Duis et tincidunt nisi. Etiam sed augue a turpis semper cursus. Proin␣
↪facilisis feugiat risus, in malesuada
               lectus posuere eget. Nullam ultricies velit purus, vel lobortis felis␣
↪commodo nec. Nam bibendum eleifend blandit.
               Vestibulum et turpis a metus euismod euismod nec sed nulla. Aliquam␣
↪iaculis, magna in posuere finibus, turpis
               .......
           </div>
           <br/>
           In ac diam sapien. Morbi viverra ante non lectus venenatis posuere.␣
↪Curabitur porttitor viverra augue sit amet
           convallis. Duis hendrerit suscipit vestibulum. Fusce fringilla convallis␣
↪eros, in vehicula nibh tempor sed.
           Fusce gravida, orci eget venenatis hendrerit, augue erat euismod magna,␣
↪nec interdum eros dolor sed ipsum.
           .....
           maximus efficitur quis velit. Phasellus et ante eget ex feugiat finibus␣
↪ullamcorper ut nisl. Sed mi
           nunc, blandit ut sem vitae, bibendum hendrerit ipsum.
       </div>
       <h1 style="margin:5pt; border-width:1pt; border-color: green">After the content
↪</h1>

       <footer>
           <h4 style="margin: 5pt; border-width: 1pt; border-color: purple">This is␣
↪the footer</h4>
```

(continues on next page)

(continued from previous page)

```
        </footer>
    </body>

    </html>
```

Here we can see that we have changed the paper orientation to landscape, set the column width to 150pt with gap of 60pt. The layout engine adjusts all content automatically within the column widths.

The second div will always be on a new column, and the 3rd div moves to a new column as it cannot fit. And the rest of the layout continues on the 3rd column until it reaches the end, and will flow onto another page.

This is the header

This is the content

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer quis orci mollis, finibus eros a, tincidunt magna. Mauris efficitur nisl lorem, vitae semper nulla convallis id. Nam dignissim rutrum mollis. Fusce imperdiet fringilla augue non venenatis. Mauris dictum velit augue, ut iaculis risus pulvinar vitae. Aliquam id pretium sem. Pellentesque vel tellus risus. Etiam dolor neque, auctor id convallis hendrerit, tincidunt at sem. Integer finibus congue turpis eu feugiat. Nullam non ultrices enim.

In ac diam sapien. Morbi viverra ante non lectus venenatis posuere. Curabitur porttitor viverra augue sit amet convallis. Duis hendrerit suscipit vestibulum. Fusce fringilla convallis eros, in vehicula nibh tempor sed. Fusce gravida, orci eget venenatis hendrerit, augue erat euismod magna, nec interdum eros dolor sed ipsum. Sed erat erat, egestas vulputate odio ullamcorper, vulputate dapibus libero. Vestibulum pharetra tincidunt nisl, facilisis cursus tortor luctus ac. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. In auctor quam vel nibh accumsan iaculis. Duis cursus tellus nec erat aliquet, id rutrum elit maximus. Suspendisse non ultricies mi. Integer efficitur sapien lectus, non laoreet tellus dictum vel. Maecenas vitae vehicula mauris. Aenean egestas et neque sit amet pulvinar.

Duis et tincidunt nisi. Etiam sed augue a turpis semper cursus. Proin facilisis feugiat risus, in malesuada lectus posuere eget. Nullam ultricies velit purus, vel lobortis felis commodo nec. Nam bibendum eleifend blandit. Vestibulum et turpis a metus euismod euismod nec sed nulla. Aliquam iaculis, magna in posuere finibus, turpis turpis fringilla eros, id sollicitudin arcu orci quis ligula. Praesent ultrices quam id ornare tempor. Morbi lobortis tempor arcu et ullamcorper. Phasellus efficitur semper iaculis. Nulla aliquet ante non magna dapibus gravida. Nunc tempus urna et lectus rhoncus, id ullamcorper neque ornare. Integer gravida dui pretium, accumsan tortor vel, cursus neque. Sed rhoncus, justo eget tristique fringilla, sem erat volutpat quam, ac consectetur nibh mi at neque.

In ac diam sapien. Morbi viverra ante non lectus venenatis posuere. Curabitur porttitor viverra augue sit amet convallis. Duis hendrerit suscipit vestibulum. Fusce fringilla convallis eros, in vehicula nibh tempor sed. Fusce gravida, orci eget venenatis hendrerit, augue erat euismod magna, nec interdum eros dolor sed ipsum. Sed erat erat, egestas vulputate odio ullamcorper, vulputate dapibus libero. Vestibulum pharetra tincidunt nisl, facilisis cursus tortor luctus ac. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. In auctor quam vel nibh accumsan iaculis. Duis cursus tellus nec erat aliquet, id rutrum elit

This is the footer

This is the header

maximus. Suspendisse non ultricies mi. Integer efficitur sapien lectus, non laoreet tellus dictum vel. Maecenas vitae vehicula mauris. Aenean egestas et neque sit amet pulvinar.

Maecenas vitae vehicula mauris. Aenean egestas et neque sit amet pulvinar. Phasellus ultrices congue semper. Praesent ultrices orci ipsum. Maecenas suscipit tellus elit, non ullamcorper nulla blandit sed. Nulla eget gravida turpis, et vestibulum nunc. Nulla mollis dui eu ipsum dapibus, vel efficitur lectus aliquam. Nullam efficitur, dui a maximus ullamcorper, quam nisi imperdiet sapien, ac venenatis diam lectus a metus. Fusce in lorem viverra, suscipit dui et, laoreet metus. Quisque maximus libero sed libero semper porttitor. Ut tincidunt venenatis ligula at viverra. Phasellus bibendum egestas nibh ac consequat. Phasellus quis ante eu leo tempor maximus efficitur quis velit. Phasellus et ante eget ex feugiat finibus ullamcorper ut nisl. Sed mi nunc, blandit ut sem vitae, bibendum hendrerit ipsum.

After the content

This is the footer

Note: As can be seen in the above image, scryber does not balance columns across the page (matching height). We may look to support this, but the min-height, max-height and breaks can be used to maintain the structure.

10.8.4 Images and Shapes in columns

As with *Sizing your content*, images and shapes that do not have an explicit size, take their natural width up to the size of the container.

This also applies to columns. If an image is too wide for the column it will be proportionally resized to fit within the column. Any content can be placed in a column.

10.8.5 Nested containers and columns

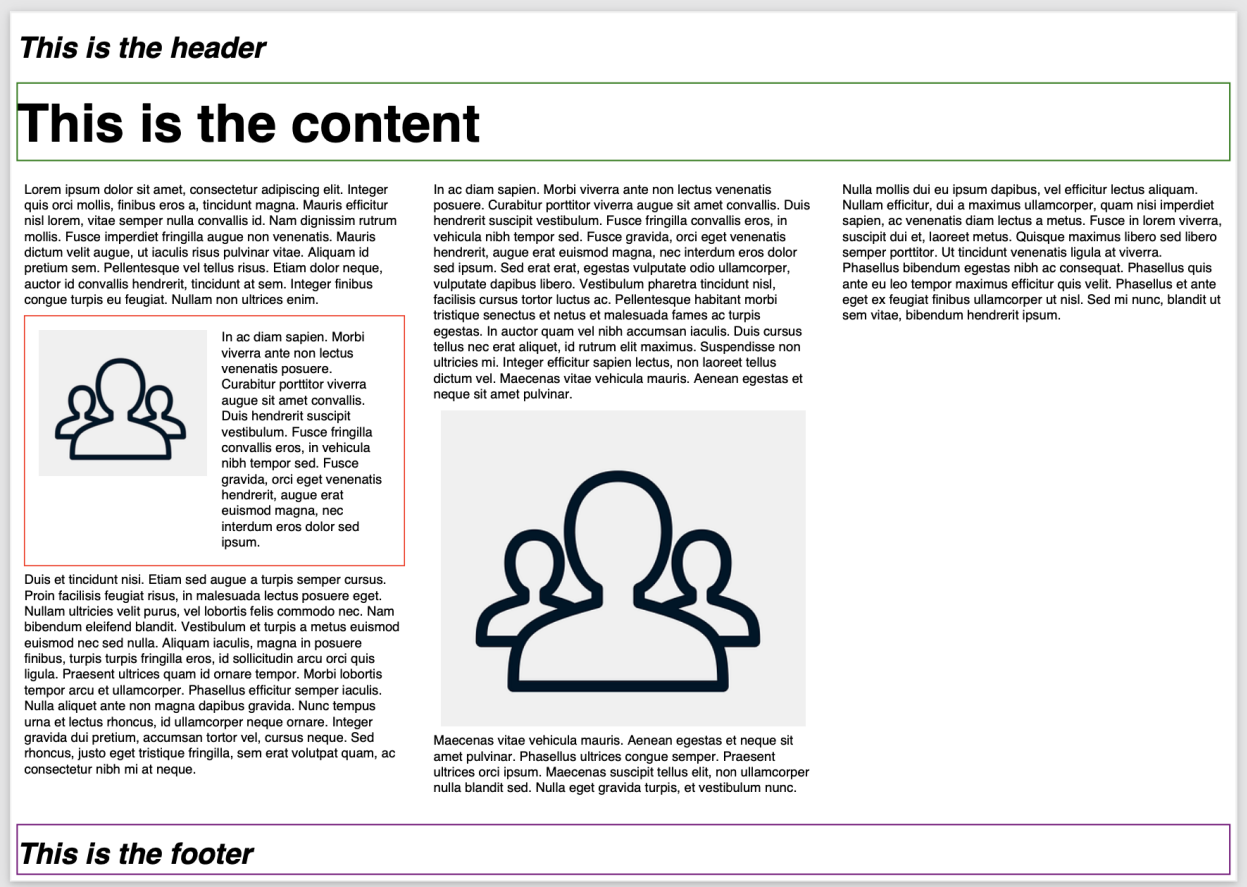
Scryber fully supports nested columns whether that be at the page or multiple container level. Again mixed content can be used within the columns, and the content will flow as normal.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css" >

        @page {
        size: A4 Landscape;
        }

    </style>
</head>
<body>
    <header>
        <h4 style='margin: 5pt; border-bottom-width: 1pt; border-bottom-color:aqua'>
↪This is the header</h4>
    </header>
    <h1 style='margin:5pt;border-width:1pt; border-color:green'>This is the content</
↪h1>
    <!-- Set a section to not break on the first page -->
    <div style="column-count: 3; column-gap:20pt; font-size:12px; padding:10pt;">
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer quis orci␣
↪mollis, finibus eros a,
        tincidunt magna. Mauris efficitur nisl lorem, vitae semper nulla convallis id.
↪ Nam dignissim rutrum
        .....
        <div style="column-count:2; border:solid 1px red; padding:10pt; margin:5pt␣
↪0pt;">
            <img src="./images/group.png" style="break-after:always;" />
            In ac diam sapien. Morbi viverra ante non lectus venenatis posuere.␣
↪Curabitur porttitor viverra augue sit amet
            convallis. Duis hendrerit suscipit vestibulum. Fusce fringilla convallis␣
↪eros, in vehicula nibh tempor sed.
            Fusce gravida, orci eget venenatis hendrerit, augue erat euismod magna,␣
↪nec interdum eros dolor sed ipsum.
```

(continues on next page)

(continued from previous page)

```
        </div>
        Duis et tincidunt nisi. Etiam sed augue a turpis semper cursus. Proin␣
↪facilisis feugiat risus, in malesuada
        lectus posuere eget. Nullam ultricies velit purus, vel lobortis felis commodo␣
↪nec. Nam bibendum eleifend blandit.
        ......
        elit maximus. Suspendisse non ultricies mi. Integer efficitur sapien lectus,␣
↪non laoreet tellus dictum vel.
        Maecenas vitae vehicula mauris. Aenean egestas et neque sit amet pulvinar.<br␣
↪/>
        <img src="./images/group.png" style="margin:5pt;" />
        Maecenas vitae vehicula mauris. Aenean egestas et neque sit amet pulvinar.
        .......
        nunc, blandit ut sem vitae, bibendum hendrerit ipsum.
    </div>
    <h1 style="margin:5pt; border-width:1pt; border-color: green">After the content</
↪h1>

    <footer>
        <h4 style="margin: 5pt; border-width: 1pt; border-color: purple">This is the␣
↪footer</h4>
    </footer>
</body>

</html>
```



10.9 Sizing your content

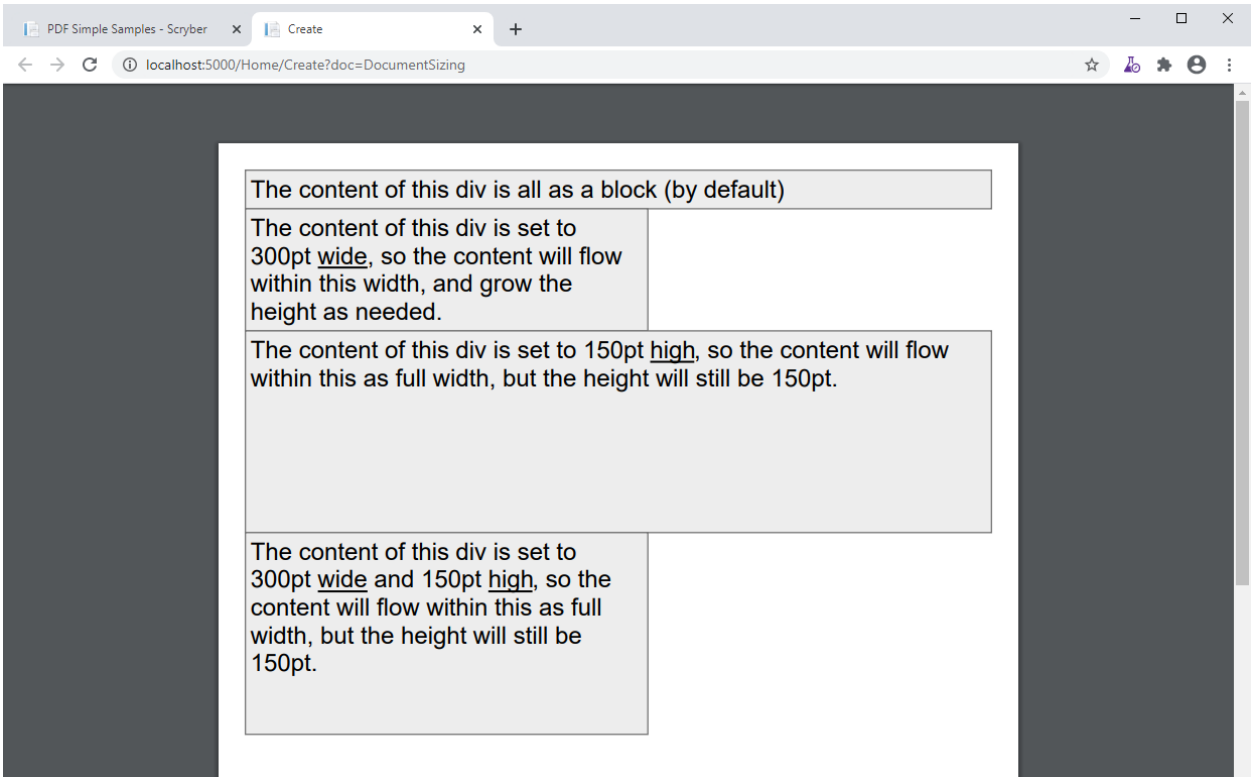
Scryber has an intelligent layout engine. By default eveything will be laid out as per the flowing layout of the document Pages and columns. Each component, be it block level or inline will have a position next to its siblings and move and following content along in the document. If the content comes to the end of the page and cannot be fitted, then if allowed, it will be moved to the next page.

However it is very easy to size and position (see *Positioning your content*) the content. All measusements are using the scryber unit and thickness (see *Units and measures* for more on the use of measurements and dimensions).

10.9.1 Width and Height

All block components support an explicit width and / or height value. If it's width is set, then any full-width style will be ignored.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css" >

        .bordered {
            border: solid 1px #777;
            background-color: #EEE;
            padding:4pt;
        }

    </style>
</head>
<body>
    <div class="bordered">
        The content of this div is all as a block (by default)
    </div>

    <div class="bordered" style="width:300pt">
        The content of this div is set to 300pt <u>wide</u>, so the content will flow␣
↪within this width,
        and grow the height as needed.
    </div>

    <div class="bordered" style="height:150pt">
        The content of this div is set to 150pt <u>high</u>, so the content will flow␣
↪within this
        as full width, but the height will still be 150pt.
    </div>

    <div class="bordered" style="width:300pt; height:150pt">
        The content of this div is set to 300pt <u>wide</u> and 150pt <u>high</u>, so␣
↪the content will flow within this
        as full width, but the height will still be 150pt.
    </div>
</body>

</html>
```



10.9.2 Images with width and height

Scryber handles the sizing of images based on the natural size of the image. If no explicit size or positioning is provided, then it will be rendered at the native size for the image.

If the available space within the container is not sufficuent to hold the image at it's natural size, then the image render size will be reduced proportionally to fit the space available.

If either a width **or** height is assigned, then this will be used to proportionally resize the image to that height or width.

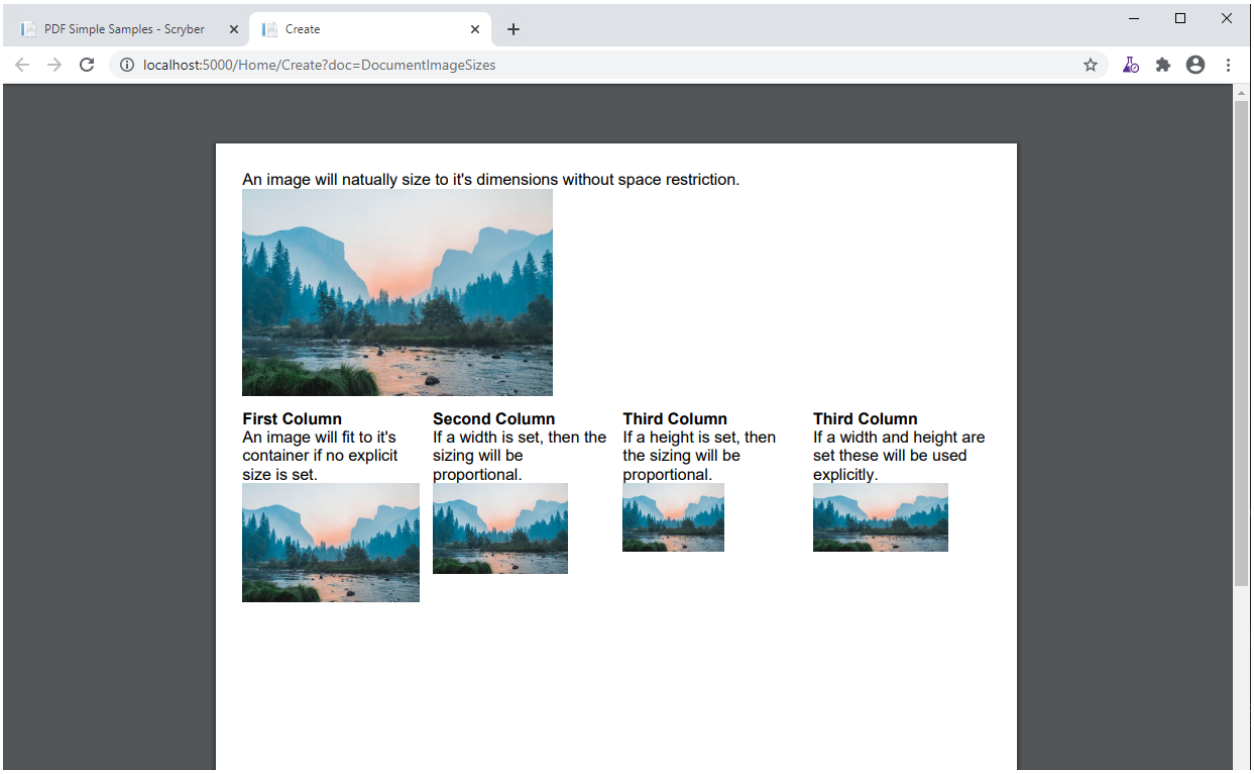
If both a width **and** height are assigned, then they will both be used to fit the image to that space. No matter what the originals' proportions are.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css" >

        body{ padding: 20pt;}

        .bordered { font-size:14pt; }

        .columns{ column-count: 4; }

        .columns div.bordered{ break-after:always; }

    </style>
```

(continues on next page)

(continued from previous page)

```
</head>
<body>
    <div class="bordered" style="margin:30pt;">
        An image will natually size to it's dimensions without space restriction.
        <img src="./images/landscape.jpg" />
    </div>
    <div class="columns" style="column-count: 4">
        <div class="bordered">
            <b>First Column</b><br />
            An image will fit to it's container if no explicit size is set.
            <img src="./images/landscape.jpg" />
        </div>
        <div class="bordered">
            <b>Second Column</b><br />
            If a width is set, then the sizing will be proportional.
            <img src="./images/landscape.jpg" style="width:100pt;" />
        </div>

        <div class="bordered" >
            <b>Third Column</b><br />
            If a height is set, then the sizing will be proportional.
            <img src="./images/landscape.jpg" style="height:50pt;" />
        </div>

        <div class="bordered" >
            <b>Fourth Column</b><br />
            If a width and height are set these will be used explicitly.
            <img src="./images/landscape.jpg" style="width:100pt; height:50pt;" />
        </div>
    </div>

    <!-- Photo by Bailey Zindel on Unsplash -->
</body>

</html>
```



10.9.3 Margins and Padding

All block level elements support padding and margins. Unlike html, scryber does not count the width of the border as part of the box dimensions (on purpose).

Dimensions can be set either directly on the component, or on a style applied to the components (see: *Styles in your template*).

The *Margin* and *Padding* style have the 4 individual properties that can also be set.

- Top
- Right
- Bottom
- and Left

If an individual side property is set, then this will override any value set on all.

The margins or padding attributes on tags can alsp be set with 1, 2 or 4 values. If only one is provided it will be applied to each. If 4 are provided, they will be applied to each individual value in the *top*, *right*, *bottom*, *left* (as per html padding). If 2 are provided the first will be applied to the top and bottom, the second to the left and right.

Note: If any margins or padding attribute is set on the component, it will override ALL values set in any style.

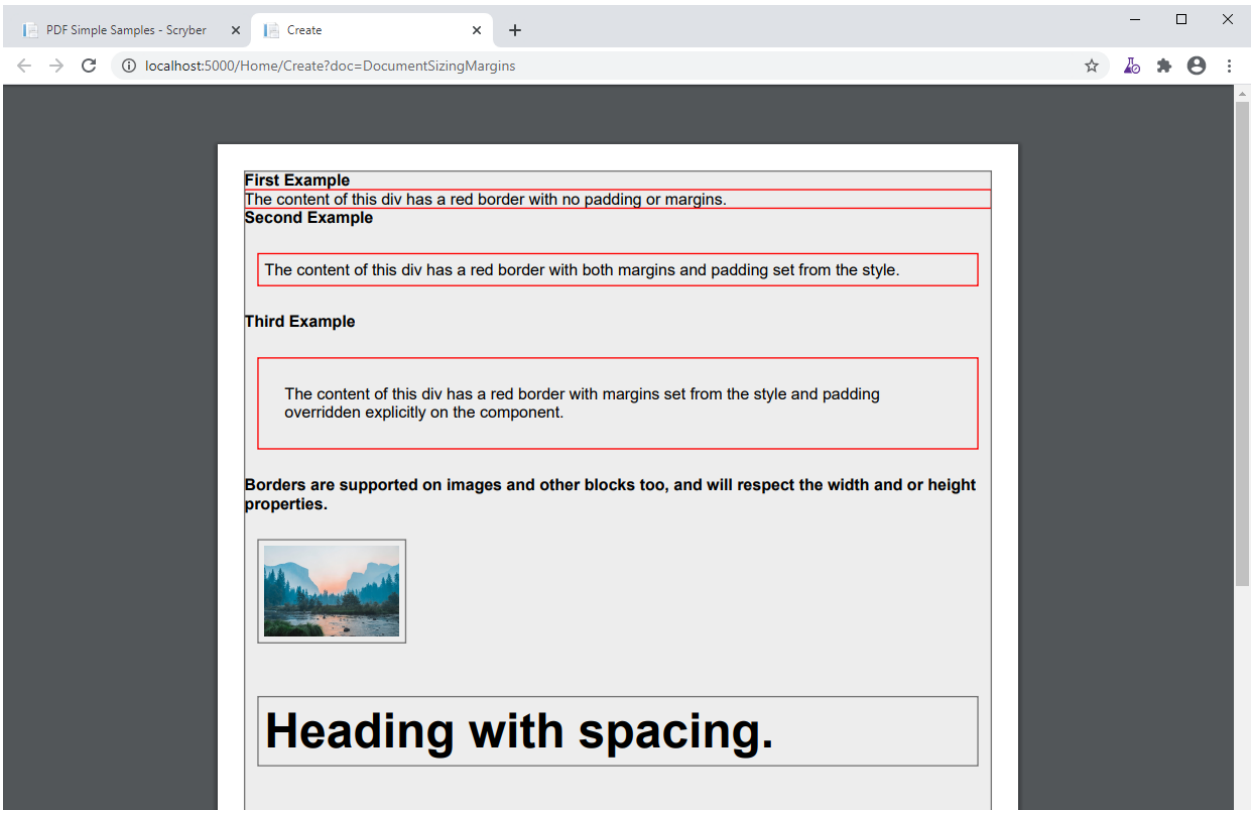
If not set then the values will be zero.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
```

(continues on next page)

(continued from previous page)

```
<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">

        body {
            margin: 20pt;
            font-size:12pt;
        }

        .bordered {
            border-style: solid;
            border-width: 1pt;
            border-color: #777;
            background-color: #EEE;
        }

        .red {
            border-color: #F00;
        }

        .spaced {
            margin: 20pt;
            margin-left: 10pt;
            margin-right: 10pt;
            padding: 5pt;
        }
    </style>
</head>
<body class="bordered">

    <b>First Example</b>
    <div class="bordered red">
        The content of this div has a red border with no padding or margins.
    </div>

    <b>Second Example</b>
    <div class="bordered red spaced">
        The content of this div has a red border with both margins and padding set␣
↪from the style.
    </div>

    <b>Third Example</b>
    <div class="bordered red spaced" style='padding:20pt;'>
        The content of this div has a red border with margins set from the style and␣
↪padding overridden explicitly on the component.
    </div>

    <b>Borders are supported on images and other blocks too, and will respect the␣
↪width and or height properties.</b>
    <img src="./Images/landscape.jpg" class="bordered spaced" style="width:100pt" />
    <h1 class="bordered spaced">Heading with spacing.</h1>

</body>
</html>
```



10.9.4 Minimum and Maximum size

Along with the use of width and height, scryber also supports the use of minimum height/width and maximum height/width.

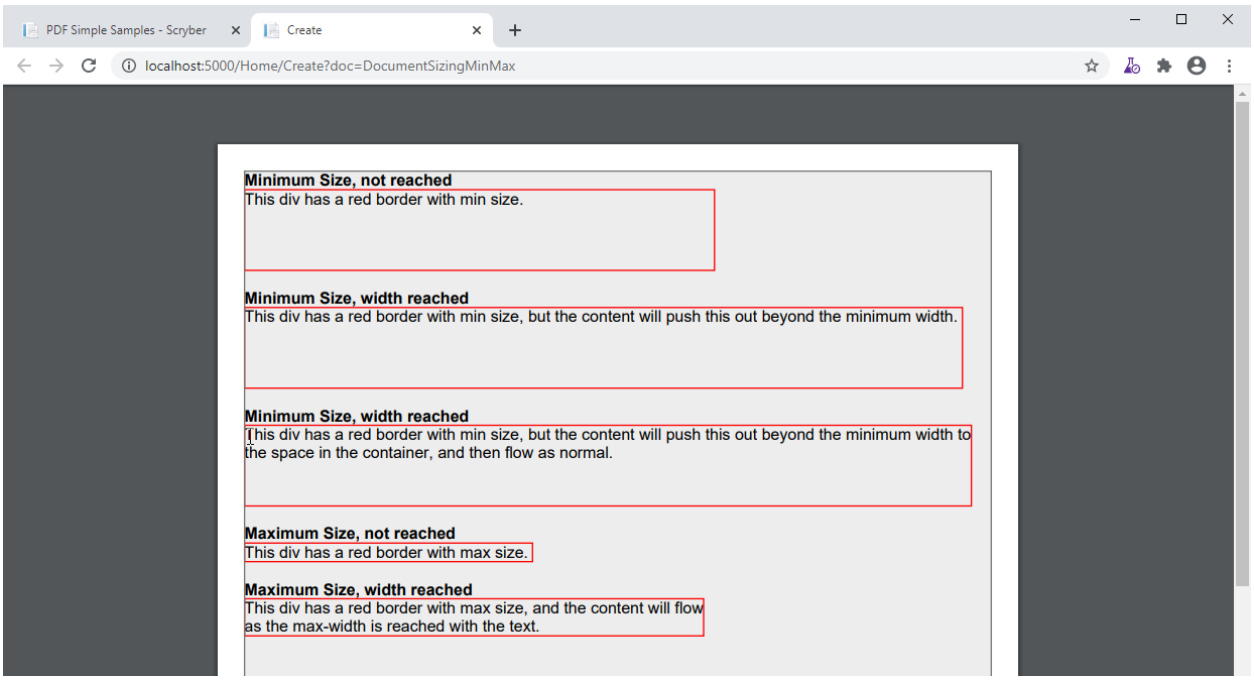
As you might expect, the minimum will ensure that a container is at least as big as the specified value, and that the maximum will ensure the content, never grows beyond that specified value.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">

        body {
            margin: 20pt;
            font-size:12pt;
        }

        .bordered {
            border-style: solid;
            border-width: 1pt;
            border-color: #777;
            background-color: #EEE;
        }
```

(continues on next page)

(continued from previous page)

```
        .red {
            border-color: #F00;
        }

        .spaced {
            margin: 20pt;
            margin-left: 10pt;
            margin-right: 10pt;
            padding: 5pt;
        }

        .sized{
            max-height:60pt;
            max-width:350pt;
        }
    </style>
</head>
<body class="bordered">
    <br />
    <b>Minimum Size, not reached</b>
    <div class="bordered red" style="min-height:60pt; min-width:350pt">
        This div has a red border with min size.
    </div>
    <br />
    <b>Minimum Size, width reached</b>
    <div class="bordered red" style="min-height:60pt; min-width:350pt">
        This div has a red border with min size, but the content will push this out␣
↪beyond the minimum width.
    </div>
    <br />
    <b>Minimum Size, width reached</b>
    <div class="bordered red" style="min-height:60pt; min-width: 350pt">
        This div has a red border with min size, but the content will push this out␣
↪beyond the minimum width to the
        space in the container, and then flow as normal.
    </div>
    <br />
    <b>Maximum Size, not reached</b>
    <div class="bordered red sized">
        This div has a red border with max size.
    </div>
    <br />
    <b>Maximum Size, width reached</b>
    <div class="bordered red sized">
        This div has a red border with max size, and the content will flow as the max-
↪width is reached with the text.
    </div>

</body>
</html>
```



10.9.5 Full width blocks

The div component automatically fills the available width of the region. Even if the inner content does not need it. It's effectivly set as 100% width.

If an explicit width, or max-width or min-width, is applied, the block will honour these rather than stretch to full width. This applies to the page, or a column containing the block.

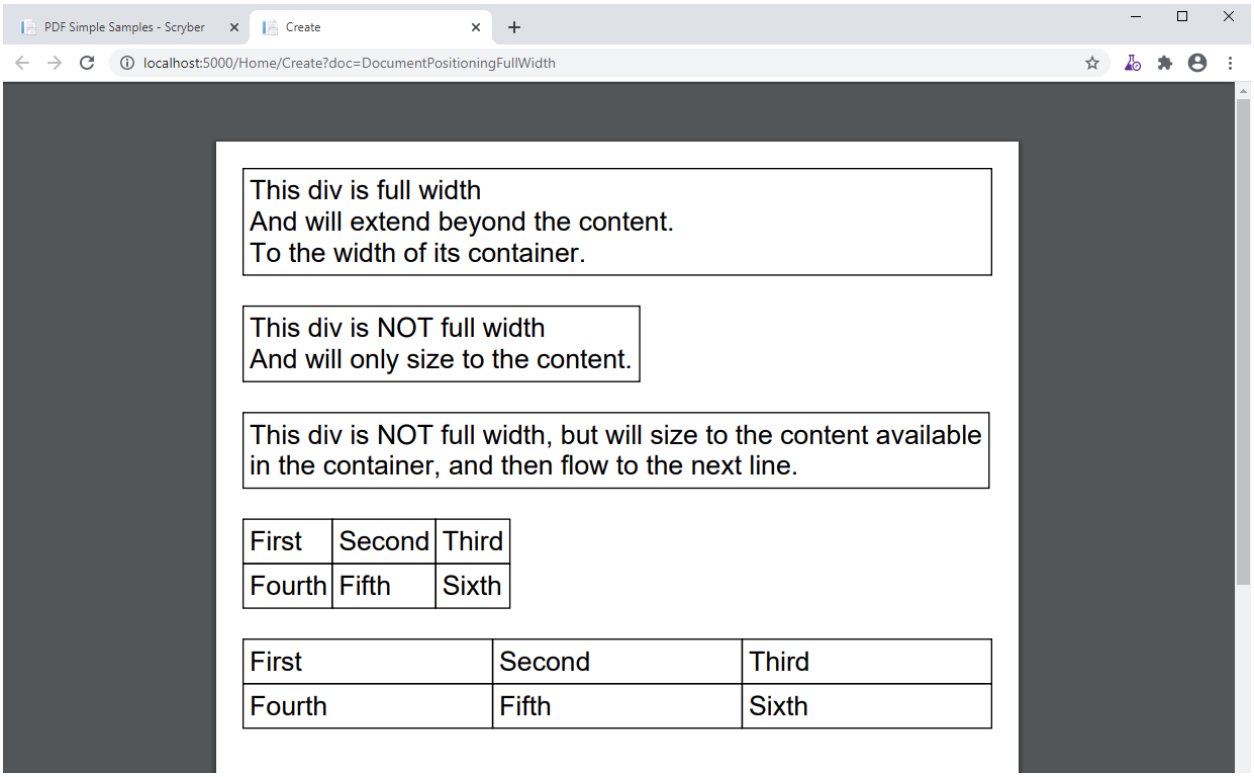
By default div's and paragraphs are set to full width. blockQuotes, tables and lists are not. If it is needed to set the width of one of these to expand to the full avaialble space, then the 100% width is supported.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<body style="margin:20pt; font-size:20pt">
    <div style="border:solid 1pt black; padding: 5pt">
        This div is full width<br />
        And will extend beyond the content.<br />
        To the width of its container.
    </div>
    <br />
    <div style="border:solid 1pt black; padding: 5pt; max-width:300pt;">
        This div is NOT full width<br />
        And will only size to the content.
    </div>
    <br />
    <div style="border:solid 1pt black; padding: 5pt; min-width:300pt;">
        This div is NOT full width,
        but will  size to the content available in the container,
        and then flow to the next line.
    </div>
```

(continues on next page)

(continued from previous page)

```
    <br />
    <table>
        <tr>
            <td>First</td>
            <td>Second</td>
            <td>Third</td>
        </tr>
        <tr>
            <td>Fourth</td>
            <td>Fifth</td>
            <td>Sixth</td>
        </tr>
    </table>
    <br />
    <!-- We set the table to 100% width so that it will expand out -->
    <table style="width:100%">
        <tr>
            <td>First</td>
            <td>Second</td>
            <td>Third</td>
        </tr>
        <tr>
            <td>Fourth</td>
            <td>Fifth</td>
            <td>Sixth</td>
        </tr>
    </table>
</body>
</html>
```



Note: Only 100% is currently supported as a relative value. 50% or 5em etc. are not supported - as pages are fixed size, the required dimensions are known and scryber uses the available space.

10.10 Positioning your content

Scryber has an intelligent layout engine. By default eveything will be laid out as per the flowing layout of the document body, sections and columns. Each component, be it block level or inline will have a position next to its siblings and move along in the document. If the content comes to the end of the page and cannot be fitted then, if allowed, it will be moved to the next page.

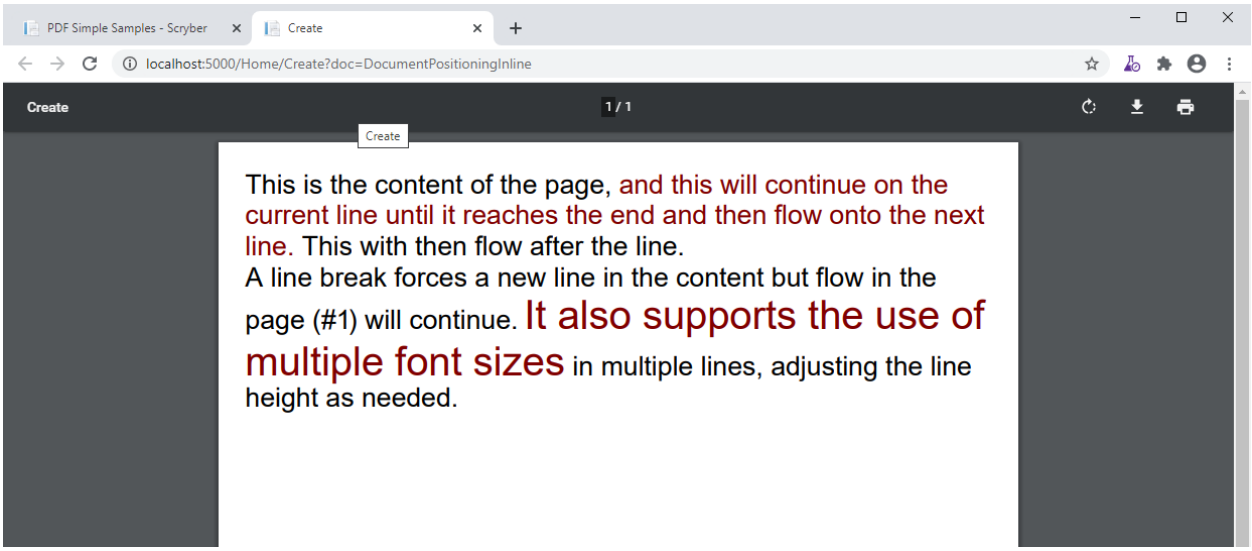
10.10.1 Inline Positioning

Inline components such as text and spans will continue on the current line, and if they do not fit all the contnet, then they will flow onto the next line (or column or page). If the content moves, so the inline content will move with the container.

Carriage returns within the content of the file are ignored by default, as per html (see *Textual Layout* if you don't want them to be.).

Examples of inline components are spans, labels, text literals, page numbers,

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<body style="margin:20pt; font-size:20pt">
    This is the content of the page,
    <span style="color:maroon">
        and this will continue on the current line until it reaches the end
        and then flow onto the next line.
    </span>
    This with then flow after the line.<br />
    A line break forces a new line in the content but flow in the page (#<page />)␣
↪will continue.
    <span style="color:maroon; font-size:30pt;">It also supports the use of multiple␣
↪font sizes</span> in multiple lines,
    adjusting the line height as needed.
</body>
</html>
```



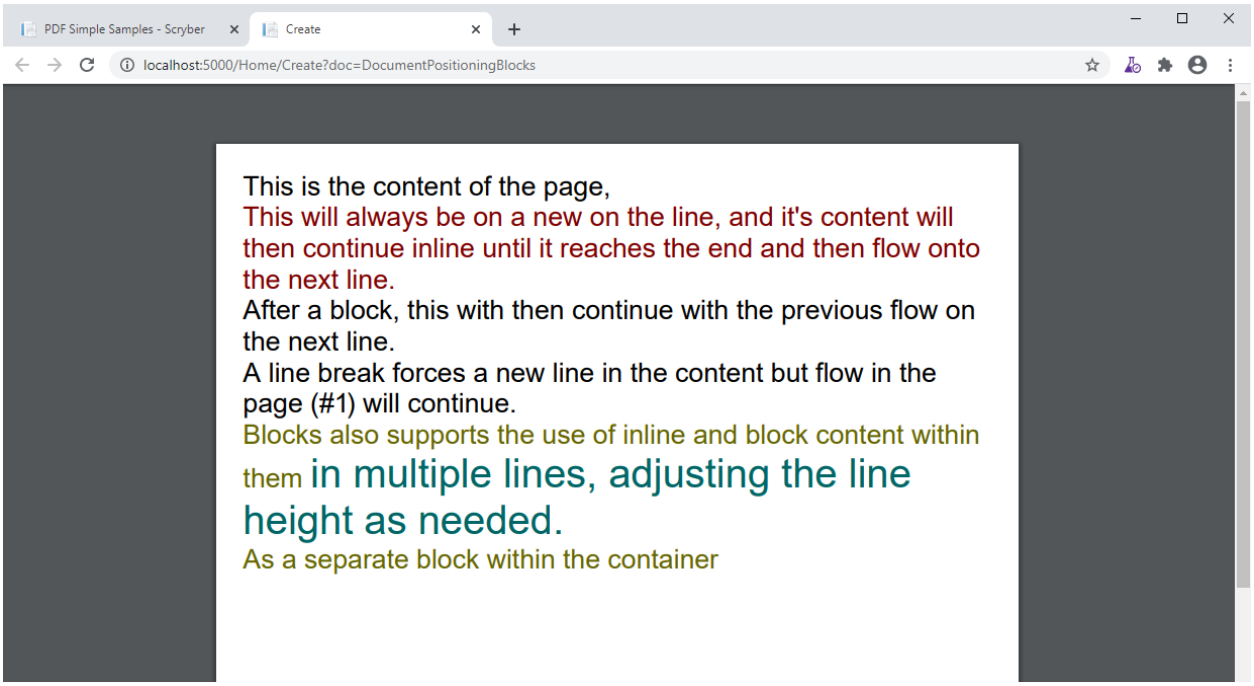
For more information on laying out textual content see *Textual Layout*

10.10.2 Block Positioning

A block starts on a new line in the content of the page. Children will be laid out within the block (unless absolutely positioned), and content after the block will also begin a new line.

Examples of blocks are Div's, Paragraphs, Tables, BlockQuotes, Headings, Images, and Shapes.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<body style="margin:20pt; font-size:20pt">
    This is the content of the page,

    <div style="color:maroon">
        This will always be on a new on the line, and it's content will then continue␣
↪inline until it reaches the end
        and then flow onto the next line.
    </div>

    After a block, this with then continue with the previous flow on the next line.
↪<br />
    A line break forces a new line in the content but flow in the page (#<page />)␣
↪will continue.

    <div style="color:#666600">
        Blocks also support the use of inline and block content within them
        <span style="color:#006666;font-size:30pt">in multiple lines, adjusting the␣
↪line height as needed.</span>
        <div>As a separate block within the container</div>
    </div>
</body>
</html>
```



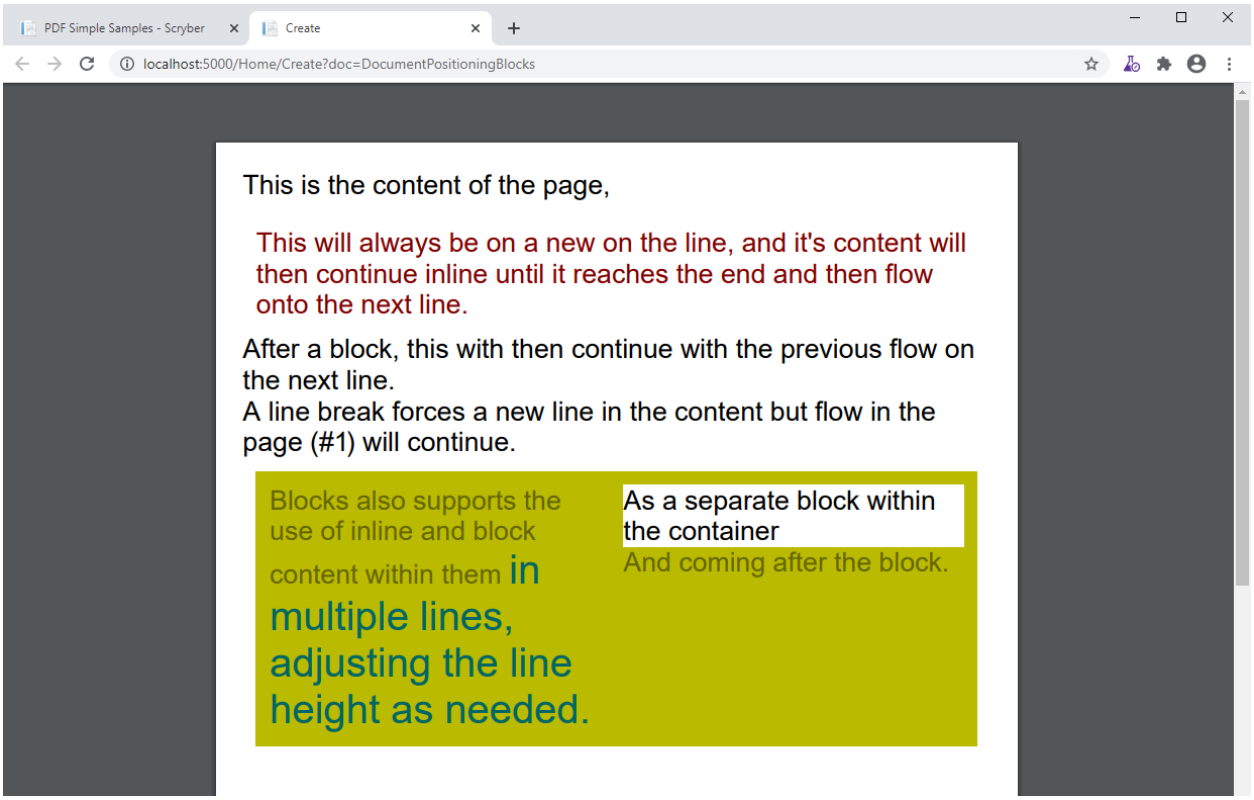
Blocks also support the use of backgrounds, borders, margins and padding. They also support *Column layout*

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<body style="margin:20pt; font-size:20pt">
    This is the content of the page,

    <div style="color:maroon; margin: 20pt 10pt 10pt 10pt">
        This will always
        be on a new on the line, and it's content will then continue inline
        until it reaches the end and then flow onto the next line.
    </div>

    After a block, this with then continue with the previous flow on the next line.
↪<br />
    A line break forces a new line in the content but flow in the page (#<page />)␣
↪will continue.

    <div style="color:#666600; background-color:#BBBB00; padding:10pt;
                margin: 10pt; column-count: 2">
        Blocks also supports the use of inline and block content within them

        <span style="color:#006666; font-size:30pt;">
            in multiple lines,
            adjusting the line height as needed.
        </span>

        <div style="color:black; background-color:white; break-before:always;">
            As a separate block within the container
        </div>
        And coming after the child block.
```

(continues on next page)

(continued from previous page)

```
    </div>
</body>
</html>
```



10.10.3 Changing the display mode

Scryber (currently) supports the following values for the display style mode:

- block
- inline
- none

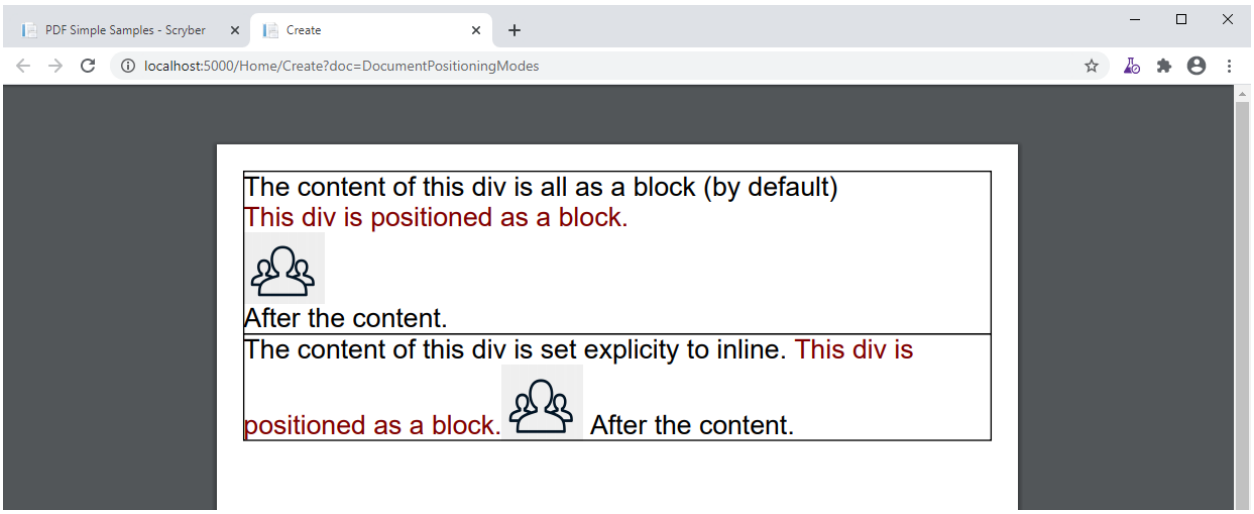
It is posible to change the default display mode for many components on the page. A span can be a block and a div can be inline. Images and shapes (see document_images and *Drawing with SVG*) also support the use of the the display mode.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<body style="margin:20pt; font-size:20pt">
    <div style="color: black; border-width: 1pt">
        The content of this div is all as a block (by default)

        <div style="color: maroon">This div is positioned as a block.</div>
```

(continues on next page)

(continued from previous page)

```
        <!-- Images are by detault displayed as blocks -->
        <img style="width:60pt" src="./Images/group.png" />

        After the content.
    </div>

    <div style="color: black; border-width: 1pt">
        The content of this div is all as a block (by default)

        <div style="color: maroon; display: inline">This div is positioned as a block.
↪</div>

        <!-- Images can be inline and will adjust the line height as needed -->
        <img style="width:60pt; display:inline" src="./Images/group.png" />

        After the content.
    </div>

    <!-- The display:none is also supported, and will not display the content. -->
    <div style="color: black; border-width: 1pt; display: none;">
        The content of this div is all as a block (by default)

        <div style="color: maroon; display: inline">This div is positioned as a block.
↪</div>

        <!-- Images are by detault displayed as blocks -->
        <img style="width:60pt; display:inline" src="./Images/group.png" />

        After the content.
    </div>
</body>
</html>
```

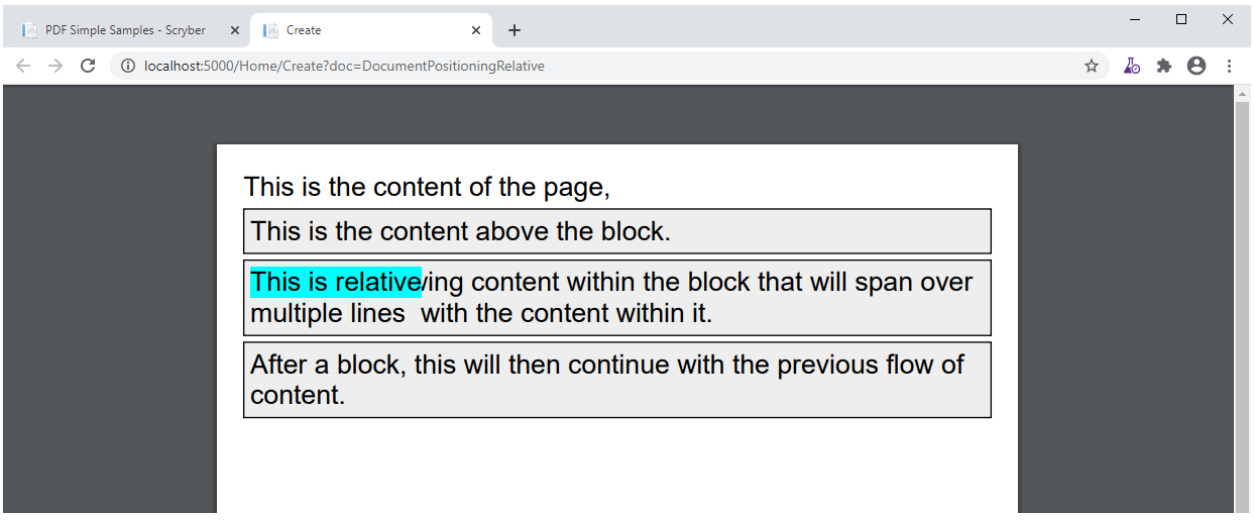


10.10.4 Relative Positioning

When you set the position-mode to Relative, it declares the position of that component relative to the block parent. The component will no longer be in the flow of any inline content, nor alter the layout of the following components.

Warning: In HTML relative has a different meaning, scryber uses the container block offsets for relative positions and the page for absolute.

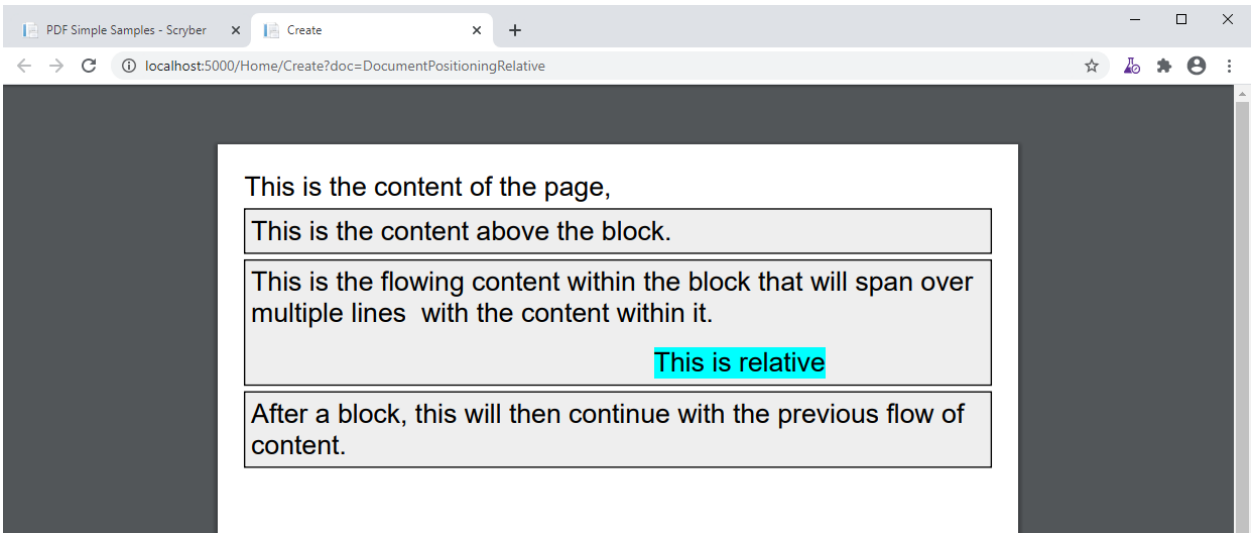
```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
    <head>
        <style type="text/css">
            .bordered{
                border: solid 1pt black;
                padding:5pt;
                background-color: #AAA;
            }
        </style>
    </head>
    <body style="margin:20pt; font-size:20pt">
        This is the content of the page,

        <div class="bordered">This is the content above the block.</div>

        <div class="bordered">
            This is the flowing content within the block that will span over multiple␣
↪lines
            <span style="position:relative; background-color:aqua">This is relative</
↪span>
            with the content within it.
        </div>

        <div class="bordered">
            After a block, this will then continue with the previous flow of content.
        </div>
    </body>
</html>
```



By default the position will be 0,0, but using the top and left values it can be altered. As soon as a left or top value are specified, the position:relative becomes inferred and is not needed.

Any parent blocks will grow to accomodate the content including any of it's relatively positioned content. And push any content after the block down.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">
        .bordered{
            border: solid 1pt black;
            padding:5pt;
            background-color: #AAA;
        }
    </style>
</head>
<body style="margin:20pt; font-size:20pt">
    This is the content of the page,

    <div class="bordered">This is the content above the block.</div>

    <div class="bordered">
        This is the flowing content within the block that will span over multiple␣
↪lines
        <span style="position:relative; top:300pt; left:60pt; background-color:aqua">
↪This is relative</span>
        with the content within it.
    </div>

    <div class="bordered">
        After a block, this will then continue with the previous flow of content.
    </div>
</body>
</html>
```



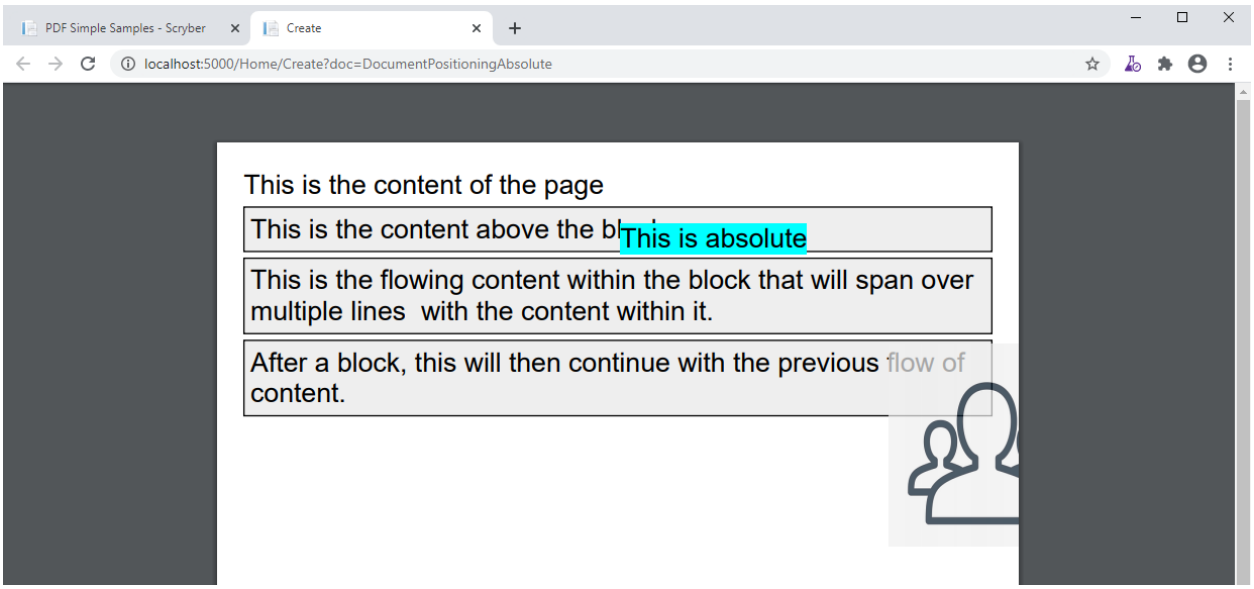
Note: By applying a position of relative the span (which is normally inline has automatically become a block and supports the background colours etc.

10.10.5 Absolute Positioning

Changing the positioning mode to Absolute makes the positioning relative to the current page being rendered. The component will no longer be in the flow of any content, nor alter the layout of following components.

The parent block will NOT grow to accomodate the content. The content within the absolutely positioned component will be flowed within the available width and height of the page, but if a size is specified, then this will be honoured over and above the page size.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">
        .bordered{
            border: solid 1pt black;
            padding:5pt;
            margin:5pt;
            background-color: #AAAAAA;
        }
    </style>
</head>
<body style="margin:20pt; font-size:20pt">
    This is the content of the page,

    <div class="bordered">This is the content above the block.</div>

    <div class="bordered">
        This is the flowing content within the block that will span over multiple␣
↪lines
        <span style="left:300pt; top:60pt; position:absolute; background-color:aqua">
            This is absolute
        </span>
        with the content within it.
    </div>

    <div class="bordered">
        After a block, this will then continue with the previous flow of content.
    </div>

    <img src="./images/group.png" style="position:absolute; top:150pt; left:500pt;␣
↪height:150pt; opacity:0.7;" />
</body>
</html>
```



10.10.6 Numeric Positioning

All content positioning is from the top left corner of the page or parent. This is a natural positioning mechanism for most cultures and developers. (unlike PDF, which is bottom left to top right).

Units of position can either be specified in

- points (1/72 of an inch) e.g *36pt*,
- inches e.g. *0.5in* or
- millimeters e.g. *12.7mm*
- pixels (1/96 of an inch) e.g. *48px*

If no units are specified then the default is points. See *Units and measures* for more information.

Note: 100% is also supported for widths to allow for the full-width capability. More support for percentage widths may be added in future.

10.10.7 Rendering Order

All relative or absolutely positioned content will be rendered to the output in the order it appears in the document. If a block is relatively positioned, it will overlay any content that preceded it, but anything coming after will be over the top.

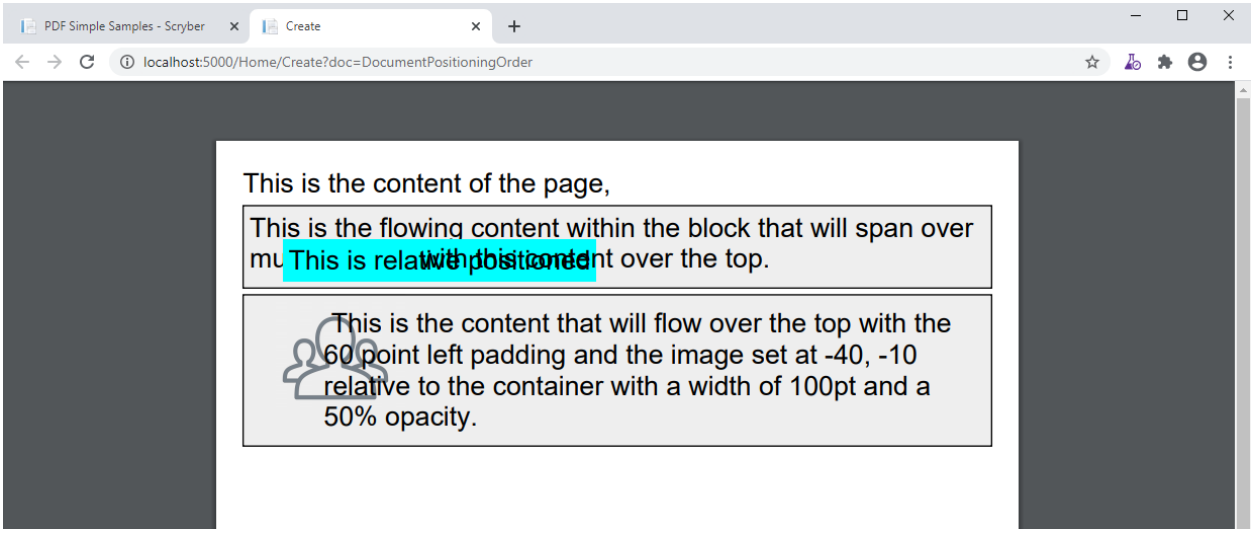
```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">
        .bordered{
```

(continues on next page)

(continued from previous page)

```
            border: solid 1pt black;
            padding:5pt;
            margin:5pt;
            background-color: #EEEEEE;
        }
    </style>
</head>
<body style="margin:20pt; font-size:20pt">
    This is the content of the page,

    <div class="bordered">This is the content above the block.</div>

    <div class="bordered">
        This is the flowing content within the block that will span over multiple␣
↪lines
        <span style="left:25pt; top:20pt; background-color:aqua; padding:4pt;">
            This is relatively positioned
        </span>
        with the content within it.
    </div>

    <div class="bordered" style="padding:10pt 10pt 10pt 60pt">
        <img src="./images/group.png"
            style="position:relative; top:-10pt; left:-40pt; width:100pt; opacity:0.5;
↪" />
        This is the content that will flow over the top with the 60 point left␣
↪padding and the
        image set at -40, -10 relative to the container with a width of 100pt
        and a 50% opacity.
    </div>

</body>
</html>
```

By using this rule interesting effects can be designed.



10.10.8 Position z-index

It's not currently supported, within scryber to specify a z-index on components. It may be supported in future.

10.10.9 Drawing Canvas

For complete control of drawing content, scryber supports svg. This can be used as drawing support for shapes and paths etc. See *Drawing with SVG* for more details.

10.11 Splitting into multiple files

For large documents or projects, it's often easier to split your templates into multiple files. These can be separate stylesheets, pages, components and the top level document.

As a converions the files should have the following extensions.

- Documents - [MyTemplate].html
- Stylesheets - [MyStyles].css
- Pages and content - [MyInnerContent].html

It just makes life easier.

10.11.1 4 file example

As an example we can split a single document into 4 files. Here we will take the top level document and reference a stylesheet, a page header component and a cover page.

10.11.2 DocumentRefs.html

At the top level is the Document - *DocumentRefs.html*

```
<?xml version="1.0" encoding="utf-8" ?>
<?scryber append-log='true' log-level='verbose' ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>My multi file document</title>
    <!-- Stylesheet links -->
    <link rel="stylesheet" media="screen" href="./css/includeScreen.css" />
    <link rel="stylesheet" media="print" href="./css/includePrint.css" />
</head>
<body style="margin:20pt; font-size:20pt">
    <header>
        <embed src="./fragments/pageheader.html" />
    </header>

    <section style="page-break-before:avoid; page-break-after:always;">
        <embed src="./fragments/coverpage.html">
    </section>
```

(continues on next page)

(continued from previous page)

```
    <div>
        <h1 class="title" >This is the second page</h1>
    </div>

</body>
</html>
```

This contains a reference to *includeScreen.css* and *includePrint.css* in the *css* folder. As the first link is specified as for screen only it will not be loaded, and only the includePrint.css will be loaded.

An embedded reference to a *PageHeader.html* in the *Fragments* folder for a standard document header, and a reference to a *CoverPage.html* in the *Fragments* folder for the cover page content.

The path references are relative to the current document (but could be absolute urls)

10.11.3 includePrint.css

This is the content of the *includePrint.css*

```
.title{
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-weight:bold;
    font-size:60pt;
    margin: 20pt;
    padding:10pt;
    text-align:center;
}

.page-head {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-weight: normal;
    font-size: 14pt;
    margin: 20pt 10pt 10pt 10pt;
    padding: 10pt;
    border-bottom:solid 1pt black;
}
```

This file declares 2 style classes that can be applied to any element with class names *title* and *page-head* For more info about styles see *Styles in your template*

10.11.4 CoverPage.html

This is the content of the *CoverPage.html*, which will be directly included in the content of the document, so should not start with the HTML of body tag, but go directly to the actual content used.

As this is intended to be the first page, and always a page, the page-break-before and page-break-after have been switched.

The namespace is important on includes, just as with top-level documents, the namespace is **critical**

```
<?xml version="1.0" encoding="utf-8" ?>
<div xmlns='http://www.w3.org/1999/xhtml' >
    <h1 class="title">Heading Page</h1>
</div>
```

Note: These are just samples and can be as complex as you like, but to be good xml it should still only have a single root.

10.11.5 PageHeader.html

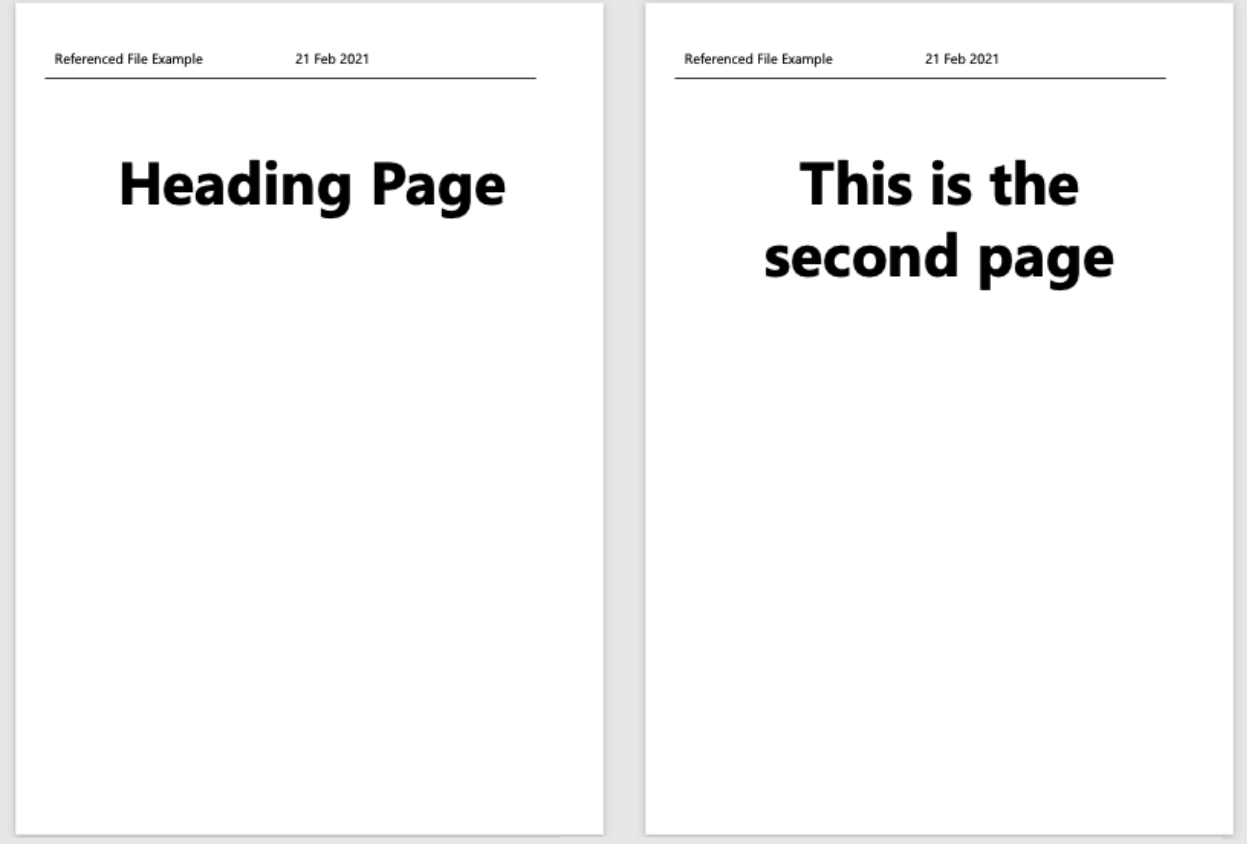
The component is referenced from link in the *DocumentRefs.pdfx*. This file is just used as the content for the header of the pages.

```
<?xml version="1.0" encoding="utf-8" ?>
<div  xmlns='http://www.w3.org/1999/xhtml' class="page-head" style="column-count:2">
    <span class="head-text" style="break-after:always;" >Referenced File Example</
↪span>
    <time date-format="dd MMM yyyy" />
</div>
```

The content could be anything, but for this time we are using it as a standard header. It has 2 columns with a title on one side and then a date label on the other.

10.11.6 Bringing it all together

These are all the files, and we just need to generate them. All being well, then when we bring it together we will get a 2 page document with consistent headers and content.



The styles are used across all content even referenced files, and the layout flows just as you would expect.

10.11.7 Circular references

Scryber will not allow circular references. i.e. files that reference either themselves, or other files that reference back to the original as it could create an infinie parsing loop.

Whilst a file can be embedded from multiple places in multiple documents, each time it will be loaded as new content. Once loaded changes to one instance will not affect any other instances loaded from that file.

10.11.8 iFrame support

Along with the embed option, scryber supports the use of iFrames with a src.

```
<iframe src='Fragments/PageHeader.html' />
```

The frame is not isolated, or independent of the main document, and styles will be transferred down into the content of the frame. This gives the wrong usage impression - but is supported as a tag element.

10.12 Dynamic content in your template

This is where it get's exciting

A document template is just that, a template. You can add any source of information to be included.

- As a discreet value
- As an object with properties
- An array or dictionary
- As a template to use.
- To make decisions on layout

And they can be used in your document in many locations

- Within the document content
- On styles and classes
- In templates and loops
- Referenced content

The notation for an binding on an attribute or content is based on the { and } with a method (@ for the built model), a colon ':', and then finally the selector.

e.g. *attribute='{@:paramName}'* for values or *attribute='{@:paramName.property[index].value}'* for objects or even in textual content.

10.12.1 Document parameters

Every Document can have parameters associated with it, and these can simply be bound to the content. This includes style and value properties as well as text, and can be set after parsing the content.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>{@:DocTitle}</title>
    <meta name="author" content="{@:DocAuthor}" />
</head>
<body style="margin:20pt; font-size:20pt">
    <header>
        <span style="{@:ThemeHeader}">{@:DocAuthor}</span>
    </header>
    <div>
        <h1 class="title" >{@:DocTitle}</h1>
        <span>{@:DocContent}</span>
    </div>

</body>
</html>
```

And the value can be set or changed at runtime

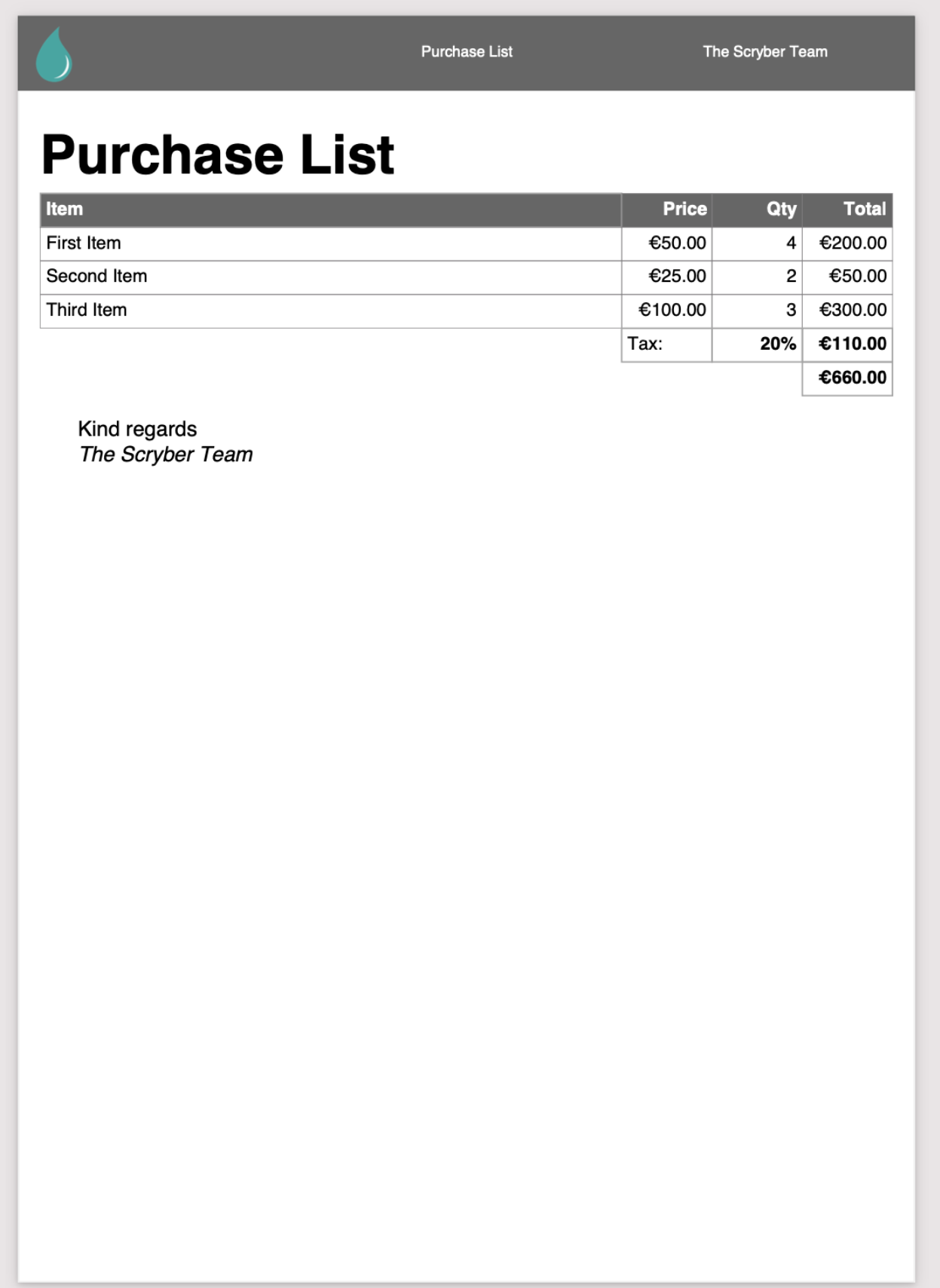
```
using (var doc = Document.ParseDocument(path))
{
    //pass paramters as needed, supporting simple values, arrays or complex classes.

    using (var stream = DocStreams.GetOutputStream("documentation.pdf"))
    {
        doc.Params["DocTitle"] = "Binding Title";
        doc.Params["DocAuthor"] = "Binding Name";
        doc.Params["ThemeHeader"] = "background-color:#EEE;padding:5pt";
        doc.Params["DocContent"] = "This is the content of the document";

        doc.SaveAsPDF(stream);
    }

}
```

And this will be used in the output.

10.12.2 Using objects

Simple values work well, but with complex entries it will start to get extremely complex. Scryber supports the standard object notations for properties arrays and dictionaries to help divide up the binding.

As a use case, we may need some purchase details.


Purchase List
The Scryber Team
Purchase List
Item
Price
Qty
Total
First Item
€50.00
4
€200.00
Second Item
€25.00
2
€50.00
Third Item
€100.00
3
€300.00
Tax:
20%
€110.00
€660.00
Kind regards
The Scryber Team

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.
↪dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>{@:Content.Title}</title>
    <meta name="author" content="{@:DocAuthor}" />
    <style>
        .header-details{ column-count:3; font-size:10pt; vertical-align:middle;}
        .header-column { break-after:always; text-align: center;}
        .header-column.logo { text-align: left; height: 40pt; max-width: 120pt;}

        .item{ border:solid 0.5px gray; }

        .list{ width:100%; font-size:12pt;}

        .item.vat, .item.price, .item.qty, .item.value{ width:60pt; text-align:right;␣
↪}
        .total, .list thead { font-weight:bold;}

        .total.empty{ border:none; }

    </style>
</head>
<body style="font-size:20pt">
    <!-- Page header with theme logo and content -->
    <header>
        <div class="header-details" style="{@:Theme.Header}">
            <img class="header-column logo" src="{@:Theme.Logo}" />
            <div class="header-column title" >{@:Content.Title}</div>
            <div class="header-column author" >{@:Content.Author}</div>
        </div>
    </header>
    <div style="margin:20px">
        <h1 class="title" >{@:Content.Title}</h1>
        <!-- A table of contents using the same theme for static headers -->
        <table class="list" >
            <thead>
                <tr style="{@:Theme.Header}">
                    <td>Item</td>
                    <td class="item price">Price</td>
                    <td class="item qty">Qty</td>
                    <td class="item value">Total</td>
                </tr>
            </thead>
            <tbody>
                <!-- and a template for the table rows looping over each of the items␣
↪-->
                <template data-bind="{@:Model.Items}">
                    <tr>
                        <!-- each one is bound with a . prefix for the current item --
↪>
                        <td class="item name">
                            <span>{@:.Item}</span>
                        </td>
                        <td class="item price" >
```

(continues on next page)

(continued from previous page)

```
                        <span>{@:.Price}</span>
                    </td>
                    <td class="item qty" >
                        <span>{@:.Quantity}</span>
                    </td>
                    <td class="item value" >
                        <span>{@:.Value}</span>
                    </td>
                </tr>
            </template>
        </tbody>
        <tfoot>
            <!-- Footer rows for the titles -->
            <tr>
                <td class="total empty" style="border:none;"></td>
                <td><span>Tax:</span></td>
                <td class="total vat" style="width:60pt; text-align:right;">
                    <span>{@:Model.Tax.Rate}</span>
                </td>
                <td class="total vat" style="width:60pt; text-align:right;">
                    <span>{@:Model.Tax.Value}</span>
                </td>
            </tr>
            <tr>
                <td colspan="3" class="total empty" style="border:none;" ></td>
                <td class="total grand" style="width:60pt; text-align:right;">
                    <span>{@:Model.Total.Value}</span>
                </td>
            </tr>
        </tfoot>
    </table>
</div>
<div id='footnote' style="padding-left:40pt; font-size: 14pt;">
    <span>Kind regards</span><br/>
    <i>{@:Content.Author}</i>
</div>
</body>
</html>
```

And with that we can bind the source into the document

```
using (var doc = Document.ParseDocument(path))
{
    //pass paramters as needed, supporting simple values, arrays or complex classes.

    using (var stream = DocStreams.GetOutputStream("documentation.pdf"))
    {
        doc.Params["Theme"] = new {
                        Header = "background-color:#666; color: white;padding:5pt",
                        Logo = "./images/ScyberLogo2_alpha_small.png"
        };

        doc.Params["Content"] = new {
            Title = "Purchase List",
            Author = "The Scryber Team"
        };
```

(continues on next page)

(continued from previous page)

```
        doc.Params["Model"] = new
        {
            Items = new[] {
                    new { Item = "First Item", Quantity = "4", Price = "€50.00",␣
↪Value = "€200.00" },
                    new { Item = "Second Item", Quantity = "2", Price = "€25.00",␣
↪Value = "€50.00" },
                    new { Item = "Third Item", Quantity = "3", Price = "€100.00",␣
↪Value = "€300.00" }
                },
            Tax = new { Rate = "20%", Value = "€110.00" },
            Total = new { Value = "€660.00" }
        };


        doc.SaveAsPDF(stream);
    }

}
```

10.12.3 Injecting content

If it is needed to inject some dynamic content within the document then it is easy to look up elements and then add the content either as html or as code.

Let's say the ask was to add an optional foot note to our Purchase list for the high demand items, and also a custom footer to the pages. We can do this in our code, without changing the template.

```
if (IsHighDemandItem())
{
    //Add the content to the footnote

    var div = doc.FindAComponentById("footnote") as Div;

    //Lets do this via conversion of dynamic xhtml into a component
    //Still needs to be valid XHTML
    var footnoteContent = "<div xmlns='http://www.w3.org/1999/xhtml'><span>Warmest␣
↪regards from all the scryber team</span><br/>" +
        "<i>" + System.Environment.UserName + "</i><br/><br/>" +
        "<b>Your order is for a high demand item. Please allow 6 weeks for delivery</
↪b></div>";

    var content = doc.ParseTemplate(doc, new System.IO.StringReader(footnoteContent))␣
↪as Component;

    //Remove the old content, as we want to
    div.Contents.Clear();
    div.Contents.Add(content);
}
```

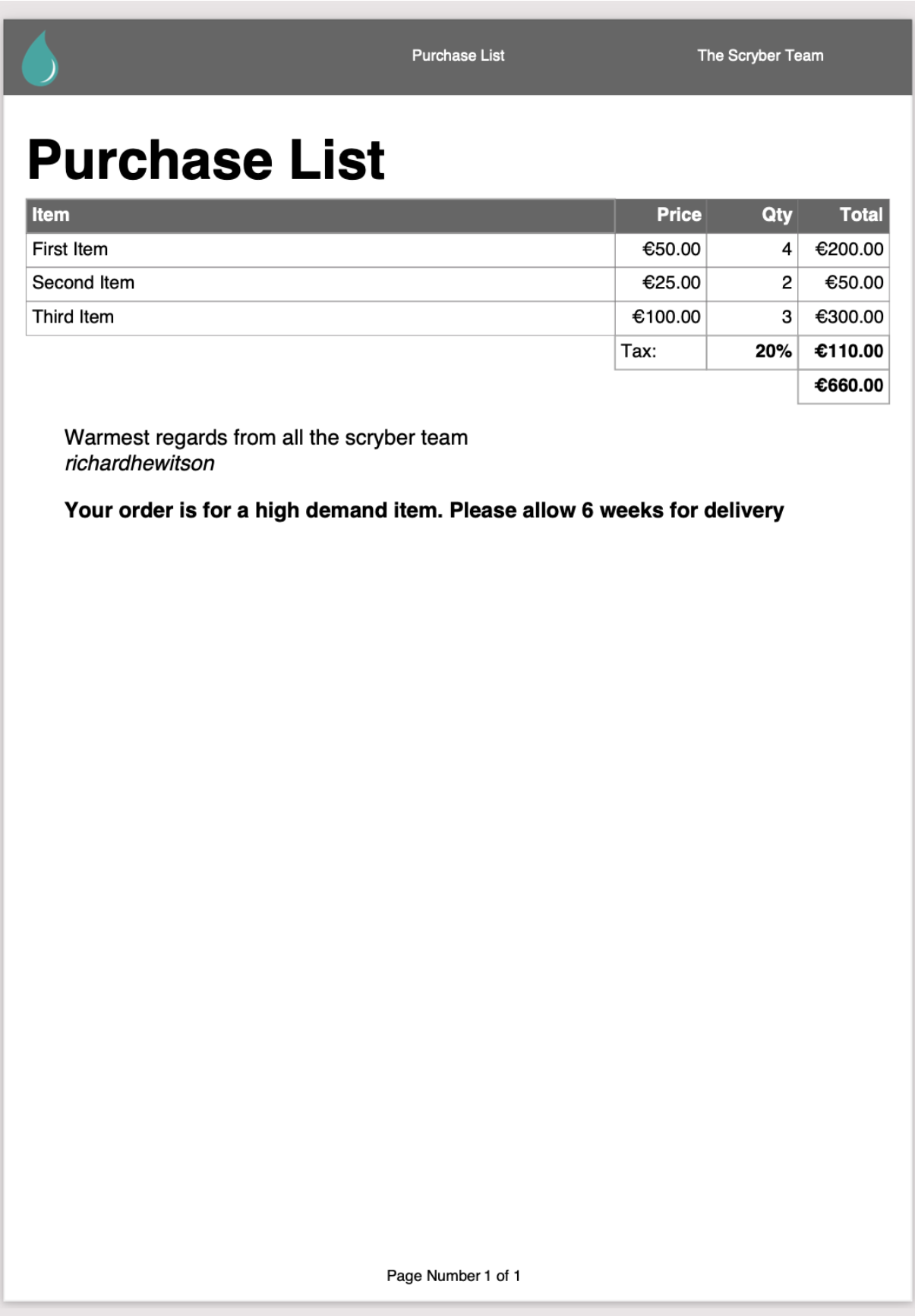
The string content is parsed, so needs to be xhtml, but then simply added to an existing div with a matching ID.

And for the footer, we use the IPDFTemplate that is used for all dynamic content building - Headers, Footers, HTML-Templates, etc.

```
//Add the custom footer
doc.Pages[0].Footer = new CustomFooter();


/// <summary>
/// Implements the IPDFTemplate for a custom footer.
/// </summary>
public class CustomFooter : IPDFTemplate
{
    /// <summary>
    /// Returns the object content (may be called multiple times).
    /// </summary>
    public IEnumerable<IPDFComponent> Instantiate(int index, IPDFComponent owner)
    {
        //Wrap it all in a div so we can set the style

        Div div = new Div() { StyleClass = "footer", FontSize = 10,
                              Padding = new PDFThickness(10),
                              HorizontalAlignment = HorizontalAlignment.Center };

        div.Contents.AddRange(new Component[]
        {
            new TextLiteral("Page Number "),
            new PageNumberLabel() { DisplayFormat = "{0} of {1}"}
        });

        return new IPDFComponent[] { div };
    }
}
```

As you can see, pretty much anything can be data bound and the output can be altered in any way using the combination of styles, declarative html content, data objects and code.

As expected this will flow with the layout of the document and maybe even onto another page.



Purchase List The Scryber Team

Purchase List

Item	Price	Qty	Total
First Item	€50.00	4	€200.00
Second Item	€25.00	2	€50.00
Third Item	€100.00	3	€300.00
	Tax:	**20%**	**€110.00**
			€660.00

Warmest regards from all the scryber team
richardhewitson

Your order is for a high demand item. Please allow 6 weeks for delivery

Page Number 1 of 1

10.13 Links in and out of documents

Within a document, it's easy to add a link to another component, another page, another document, or remote web link.

10.13.1 The Anchor Component

An 'a'nchor tag is an invisible component (although styles will be applied to content within it) that allows linking to other components. The href attribute supports 3 types of content.

- Named Action - These are the standard links to and from pages (see below).
- #id - If set, then the link is an action for a different document or Url. Effectively like the href of an anchor tag in html.
- url - A relative or remote link to a document or webpage.

The content within a link can be anything, including images; text; svg components and more. There can also be more than one component within the link.

10.13.2 Page Named Action

The simplest link type is navigational. The possible actions are (case insensitive) as follows:

- FirstPage
- PrevPage
- NextPage
- LastPage

These are self-evident in their purpose, and no other attributes need defining. It does not matter what page they are put on, they will perform the action if possible.

```
<a href='nextpage' >Next Page Link</a>
```

For example we can create a navigation header.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>Navigation Links</title>
    <meta name="author" content="Scryber" />
    <style type="text/css">
        /* Or Header Style with 4 columns */
        .header{
            background-color: #EEE;
            column-count:4;
            text-align:center;
        }
        /* Our Link button style */
        .header a{
            display:block;
            border: solid 1pt #DDD;
```

(continues on next page)

(continued from previous page)

```
            border-radius: 4pt;
            padding:6pt;
            margin:6pt;
            break-after:always;
            text-decoration:none;
            color: #333;
            font-size:10pt;

        }
        /* And our divs */
        div {
            margin: 20pt 0 0 20pt;
        }
        div.break{
            page-break-after:always;
        }

    </style>
</head>
<body style="font-size:20pt;">
    <header>
        <!-- navigation on the header -->
        <nav class="header">
            <a href="FirstPage">First Page</a>
            <a href="PrevPage">Prev Page</a>
            <a href="NextPage">Next Page</a>
            <a href="LastPage">Last Page</a>
        </nav>
    </header>
    <!-- Each on it's own page -->
    <div class="break">First</div>
    <div class="break">Second</div>
    <div class="break">Third</div>
    <div class="break">Fourth</div>
    <div>Fifth</div>
</body>
</html>
```

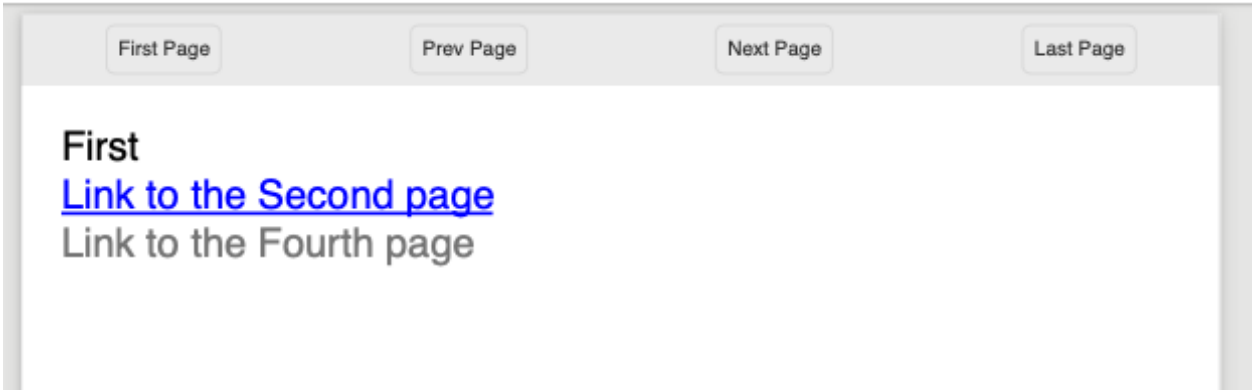

First Page
Prev Page
Next Page
Last Page
First
First Page
Prev Page
Second

10.13.3 Linking within documents

When navigating around the documment, scryber supports the direct linking to a specific page or component using the id being referenced attribute. Prefix with a # to identify it is an element witin the document.

By default anchor links will be underlined and in blue. But can be styled as needed.

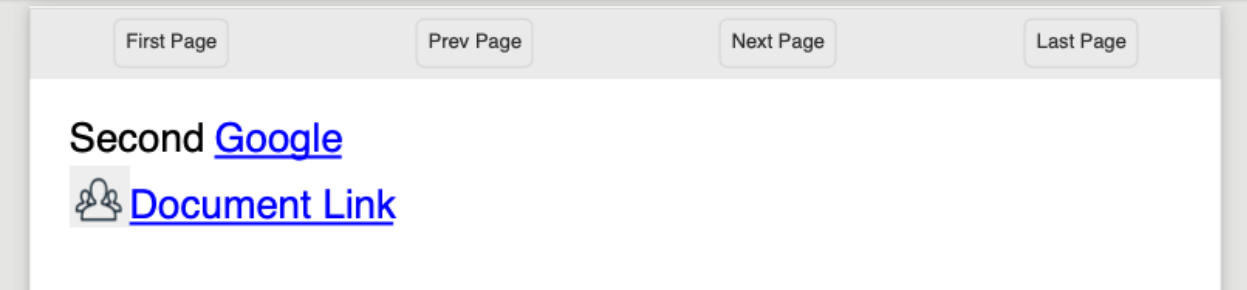
```
<?xml version="1.0" encoding="utf-8" ?>

<div id="first" class="break">
    First <br />
    <a href="#second" >Link to the Second page</a>
    <br />
    <a href="#fourth" style="text-decoration:none; color:gray;" >Link to the Fourth␣
↪page</a>
</div>
<div id="second" class="break">Second</div>
<div id="third" class="break">Third</div>
<div id="fourth" class="break">
    Fourth <br />
    <a href="#first" >Link to the first page</a>
</div>
<div>Fifth</div>
```



10.13.4 External Links to Urls

Using the href attribute a remote link can be made to any url or local document. If it's not one of the other type it will be assumed to be a link. Links can also contain images or any other content, and can use the target='_blank' to open in a new tab.

```
 <!-- A web link to the google home page -->
<a href="https://www.google.com" target="_blank" >Google</a><br/>

<!-- a link to a local pdf that will open in a new readr tab or window -->
<a alt="Document Link" href="ReadMeSample.pdf" target="_blank" >
    <img src="./images/group.png" style="width:30pt; display:inline" />Document Link</
↪a>
```



Note: Some of the browser pdf readers do not support the full navigaional links capabilities (or allow them). Reader applications generally do.

10.14 Document fonts and text styles

PDF supports standard fonts. Scryber also supports the standard names of 'sans-serif', 'serif', 'monospace'.

Fonts available in the current operating system that the application has access to, can also be used. They are referenced by their postscript name.

Fonts can also be dynamically included with the @font-face rule.

Scryber supports the use of the font fallback in styles.

Note: scryber currently only supports True type &tm; and Open Type fonts - ttf (otf) and ttc (otc)

10.14.1 Built in fonts

The standard built in fonts with PDF readers that can be used in documents. These are as follows

- sans-serif / Helvetica - Regular, Bold, Italic and Bold Italic.
- serif / Times - Regular, Bold, Italic and Bold Italic.
- monospace / Courier - Regular, Bold, Italic and Bold Italic.

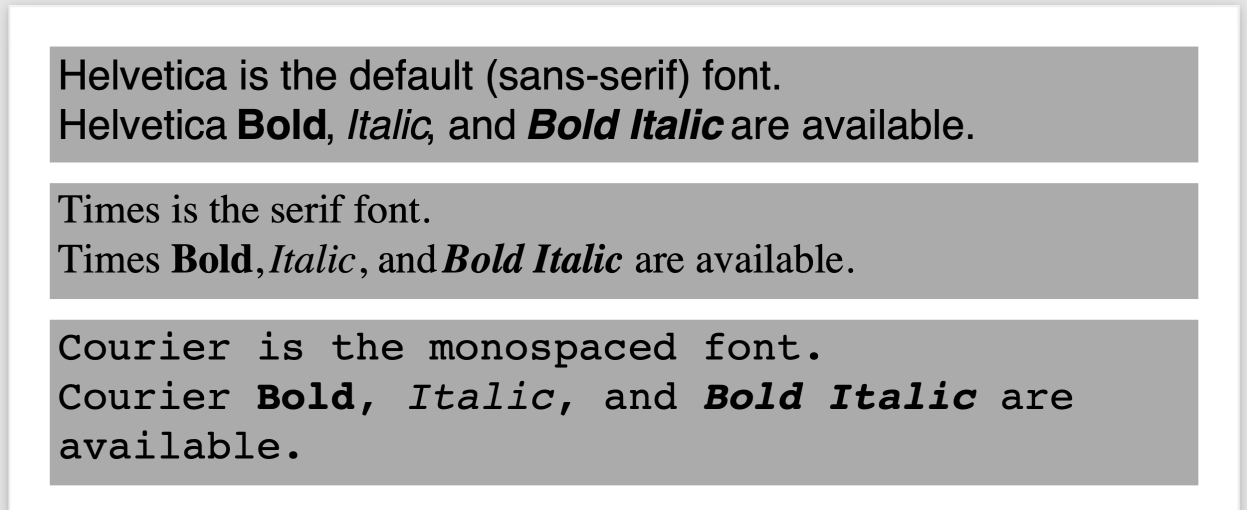
If used, then if the font is available it will be embedded, as is best practice.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>{@:Content.Title}</title>
    <meta name="author" content="{@:DocAuthor}" />
    <style type="text/css">

        .std-font{
            font-size: 20pt;
            background-color:#AAA;
            padding: 4pt;
```

(continues on next page)

(continued from previous page)

```
            margin-bottom:10pt;
        }

    </style>
</head>
<body style="padding: 20pt">
    <div id="first" class="std-font" style="font-family:sans-serif">
        Helvetica is the default (sans-serif) font.<br/>
        Helvetica <b>Bold</b>, <i>Italic</i>, and <span style="font-weight:bold; font-
↪style:italic">Bold Italic</span> are available.
    </div>

    <div id="first" class="std-font" style="font-family:serif">
        Times is the serif font.<br />
        Times <b>Bold</b>, <i>Italic</i>, and <span style="font-weight:bold; font-
↪style:italic">Bold Italic</span> are available.
    </div>

    <div id="first" class="std-font" style="font-family:monospace">
        Courier is the monospaced font.<br />
        Courier <b>Bold</b>, <i>Italic</i>, and <span style="font-weight:bold; font-
↪style:italic">Bold Italic</span> are available.
    </div>


</body>
</html>
```

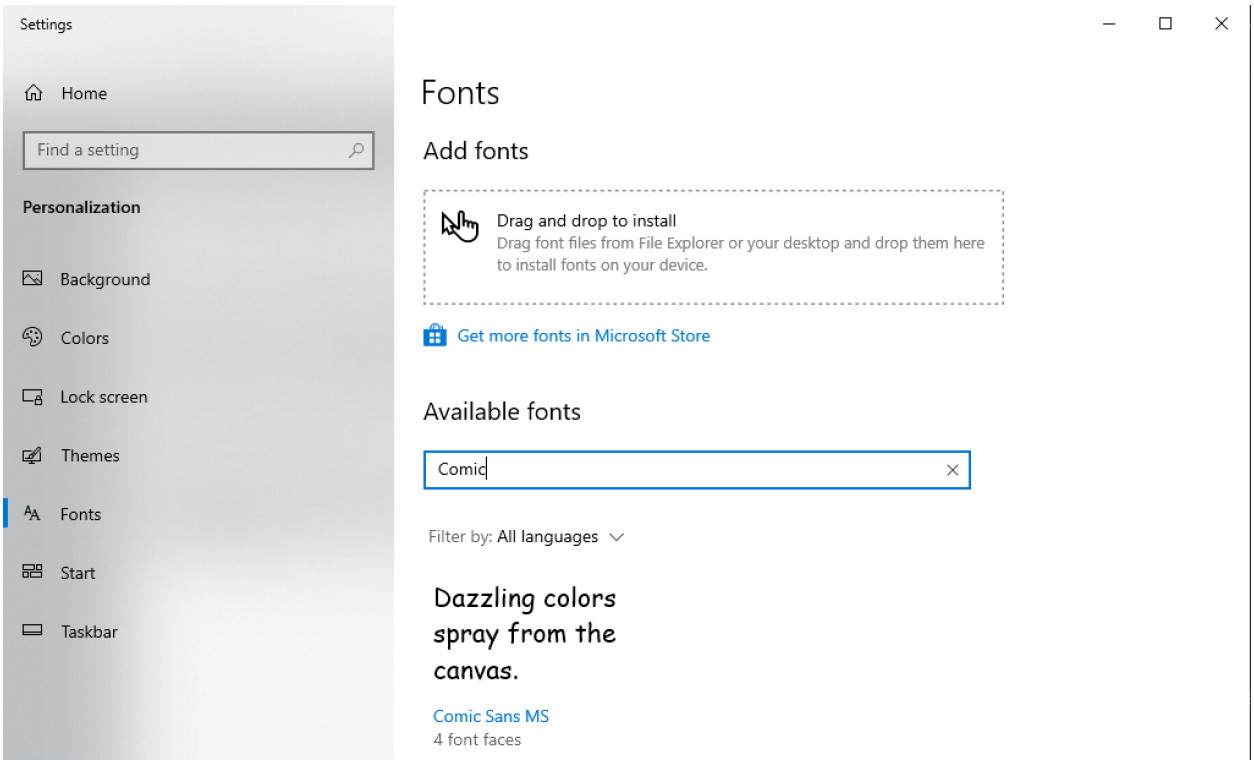


10.14.2 Using different fonts

Along with the standard fonts, scryber supports the systems fonts (the fonts in the Environment.SpecialFolder.Fonts). It does not support postscript font files but does support.

- ttf & otf - A truetype font file or opentype font file.
- ttc & otc - A truetype font collection (multiple styles) or open type collection

Fonts should be referred to by their Unicode Name, usually displayed through the font browser of the underlying operating system. Rather than the file name of the ttf or ttc file.



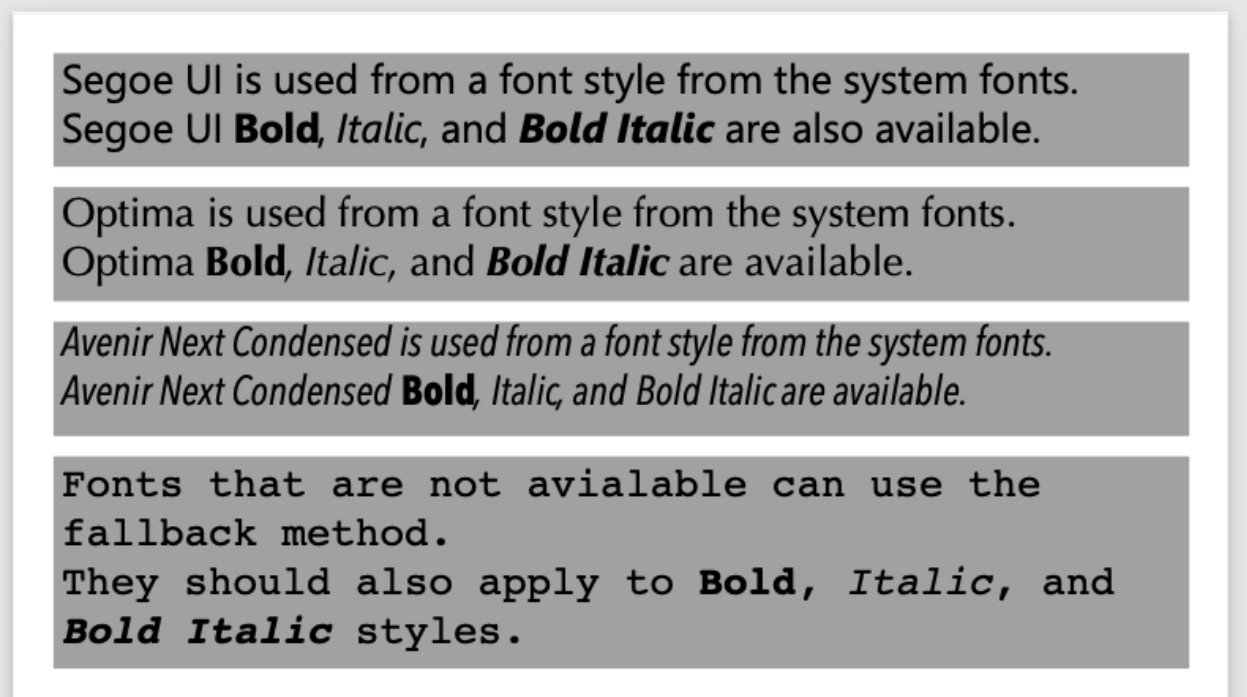
The following uses 3 different ttf fonts installed on the machine generating the document. But using the standard css font fallback if a font does not exist it can fall back to one of the known fonts

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>System Font Files</title>
    <meta name="author" content="Scryber Team" />
    <style type="text/css">

        .std-font{
            font-size: 20pt;
            background-color:#AAA;
            padding: 4pt;
            margin-bottom:10pt;
        }

        .sans {
            font-family: 'Segoe UI', sans-serif;
        }

        .serif{
            font-family: Optima, Times, Times New Roman, serif;
        }

        .avenir{
            font-family:'Avenir Next Condensed', sans-serif;
        }
```

(continues on next page)

(continued from previous page)

```
        .none {
            font-family: 'Made Up Font', monospace;
        }
    </style>
</head>
<body style="padding: 20pt">
    <div id="first" class="std-font sans">
        Segoe UI is used from a font style from the system fonts.<br />
        Segoe UI <b>Bold</b>, <i>Italic</i>, and <span style="font-weight:bold; font-
↪style:italic">Bold Italic</span> are also available.
    </div>

    <div id="first" class="std-font serif">
        Optima is used from a font style from the system fonts.<br />
        Optima <b>Bold</b>, <i>Italic</i>, and <span style="font-weight:bold; font-
↪style:italic">Bold Italic</span> are available.
    </div>

    <div id="first" class="std-font avenir">
        Avenir Next Condensed is used from a font style from the system fonts.<br />
        Avenir Next Condensed <b>Bold</b>, <i>Italic</i>, and <span style="font-
↪weight:bold; font-style:italic">Bold Italic</span> are available.
    </div>

    <div id="first" class="std-font none">
        Fonts that are not avialable can use the fallback method.<br />
        They should also apply to <b>Bold</b>, <i>Italic</i>, and <span style="font-
↪weight:bold; font-style:italic">Bold Italic</span> styles.
    </div>
</body>
</html>
```



As the font is set to inherit, all child text components will use the specified font of the parent. If the font is changed, then all children will use the new font.

Note: .woff or woff files are not currently supported, but these can be easily converted to their ttf components online. They may be supported in future.

10.14.3 Specifying the font in code

If it is needed to set the actual font in code then internally the PDFFontSelector class can be used. It has a constructor that takes the name of the font, and an overload that can be used to chain multiple selectors together.

If the language supports it, then ther is also an explict cast available to convert to a PDFFontSelector.

```
page.Style.Font.FontFamily = new PDFFontSelector("sans-serif");
page.FontFamily = new PDFFontSelector("Arial", new PDFFontSelector("sans-serif"));
page.FontFamily = (PDFFontSelector)"Arial, sans-serif";
```

10.14.4 Font face loading

Along with the standard and system installed fonts, scryber supports the importing and declaration of custom fonts from specific files.

These can either be relative to the current file, or an absolute url.

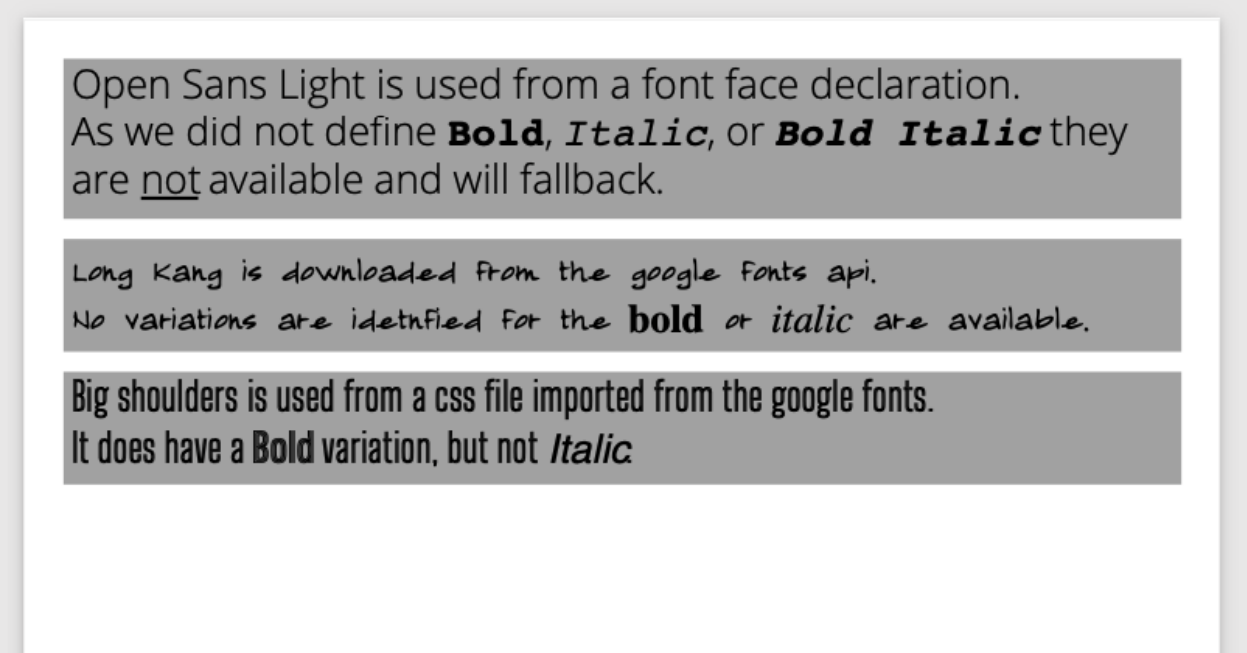
This is also a good way of specifying various weights, as scryber (currently) only supports the bold variant. It is on our list of todo's.

```
<?xml version="1.0" encoding="utf-8" ?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>Fonts loaded directly</title>
    <meta name="author" content="Scryber Team" />
    <!--
        link to google fonts API's.
      -->
    <link href="https://fonts.googleapis.com/css2?
↪family=Big+Shoulders+Inline+Display:wght@500;800&amp;display=swap" rel="stylesheet"␣
↪/>
    <style type="text/css">

        /* Open light font file on the local drive */

        @font-face {
            src: url(../../Resources/OpenSans-Light.ttf) format('truetype');
            font-family: 'Open Light';
        }

        /* Long Cang is downloaded from google but is cached locally */

        @font-face {
```

(continues on next page)

(continued from previous page)

```
            font-family: 'Long Cang';
            font-style: normal;
            font-weight: 400;
            src: url(https://fonts.gstatic.com/s/longcang/v5/LYjAdGP8kkgoTec8zkRgrQ.
↪ttf) format('truetype');
        }

        .std-font {
            font-size: 20pt;
            background-color: #AAA;
            padding: 4pt;
            margin-bottom: 10pt;
        }

        /* Setting the classes to the fonts above */

        .sans {
            font-family: 'Open Light', monospace;
        }

        .grafitti {
            font-family: 'Long Cang', serif;
        }

        .broad {
            font-family: 'Big Shoulders Inline Display', sans-serif;
        }

    </style>
</head>
<body style="padding: 20pt">
    <div id="first" class="std-font sans">
        Open Sans Light is used from a font face declaration.<br />
        As we did not define <b>Bold</b>, <i>Italic</i>, or <span style="font-
↪weight:bold; font-style:italic">Bold Italic</span> they are <u>not</u> available␣
↪and will fallback.
    </div>

    <div id="first" class="std-font grafitti">
        Long Kang is downloaded from the google fonts api.<br />
        No variations are idetnfied for the <b>bold</b> or <i>italic</i> are␣
↪available.
    </div>

    <div id="first" class="std-font broad">
        Big shoulders is used from a css file imported from the google fonts.<br />
        It does have a <b>Bold</b> variation, but not <i>Italic</i>.
    </div>

</body>
</html>
```



Warning: The link for the font css from google is not XHTML compliant. The & parameter separator should be escaped to & and the link tag closed '/>'

10.14.5 Text styles and decoration

Along with the bold and italic variants, scryber also supports underlines, strikethrough and overline text rendering features. As with HTML these are default styles, and can be altered as needed.

- **Bold**
  -
  -
  - css {font-weight:bold;}
- **italic**
  -
  -
  - css {font-style:italic;}
- **Underline**
  - <u></u>
  - <ins></ins>
  - css { text-decoration:underline;}
- **StrikeThrough**
  - <strike></strike>
  - <del></del>

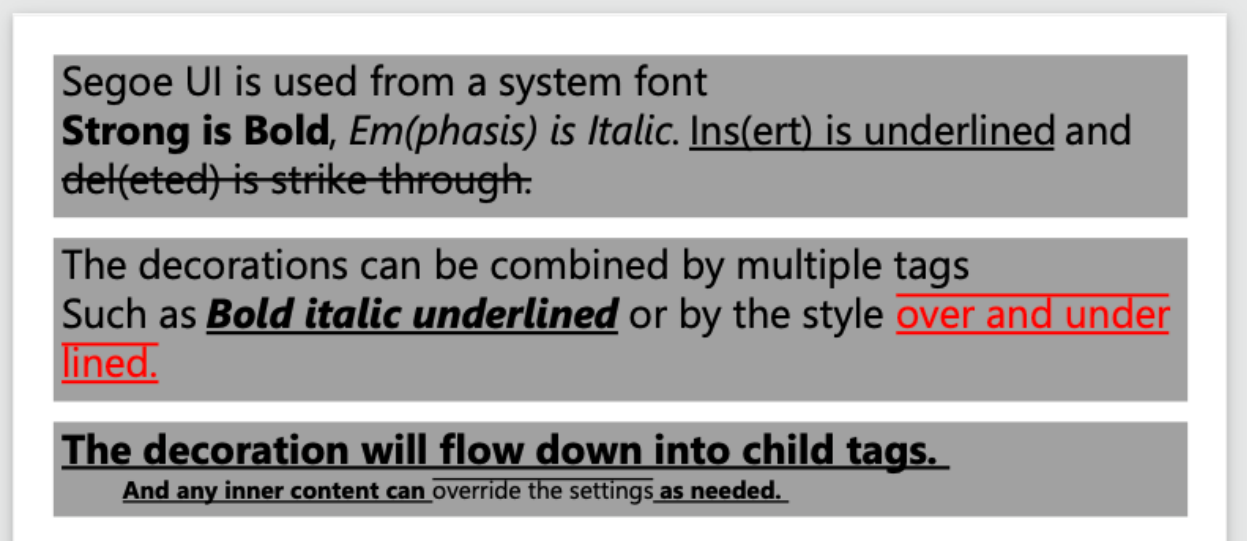
  - css { text-decoration: line-through; }

- **Overline**
  - css { text-decoration: overline; }

As with css text-decoration values can be combined e.g. 'line-through underline' , and the decorations will flow across lines.

Scryber does not (currently) support the text-decoration-color or text-decoration-style.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>Fonts decorations</title>
    <meta name="author" content="Scryber Team" />
    <style type="text/css">

        .std-font {
            font-size: 20pt;
            background-color: #AAA;
            padding: 4pt;
            margin-bottom: 10pt;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        }

        .railway{ text-decoration: overline underline; color: red;}

    </style>
</head>
<body style="padding: 20pt">
    <div id="first" class="std-font">
        Segoe UI is used from a system font<br />
        <strong>Strong is Bold</strong>, <em>Em(phasis) is Italic</em>.
        <ins>Ins(ert) is underlined</ins> and <del>del(eted) is strike through.</del>
    </div>

    <div id="first" class="std-font">
        The decorations can be combined by multiple tags<br />
        Such as <b><em><u>Bold italic underlined</u></em></b>
        or by the style <span class="railway" >over and under lined.</span>
    </div>

    <div class="std-font" style="font-weight:bold; text-decoration: underline;" >
        The decoration will flow down into child tags.
        <div style="margin:0 30pt 0 30pt; font-size:12pt">And any inner content can
            <span style="font-weight:normal; text-decoration: overline;">override the␣
↪settings</span>
            as needed.
        </div>
    </div>
</body>
</html>
```



10.14.6 Line height (leading)

The leading is the height of the lines including ascenders and descenders. The default is set by the font (usually about 120% of the font size), but can be manually adjusted as needed.

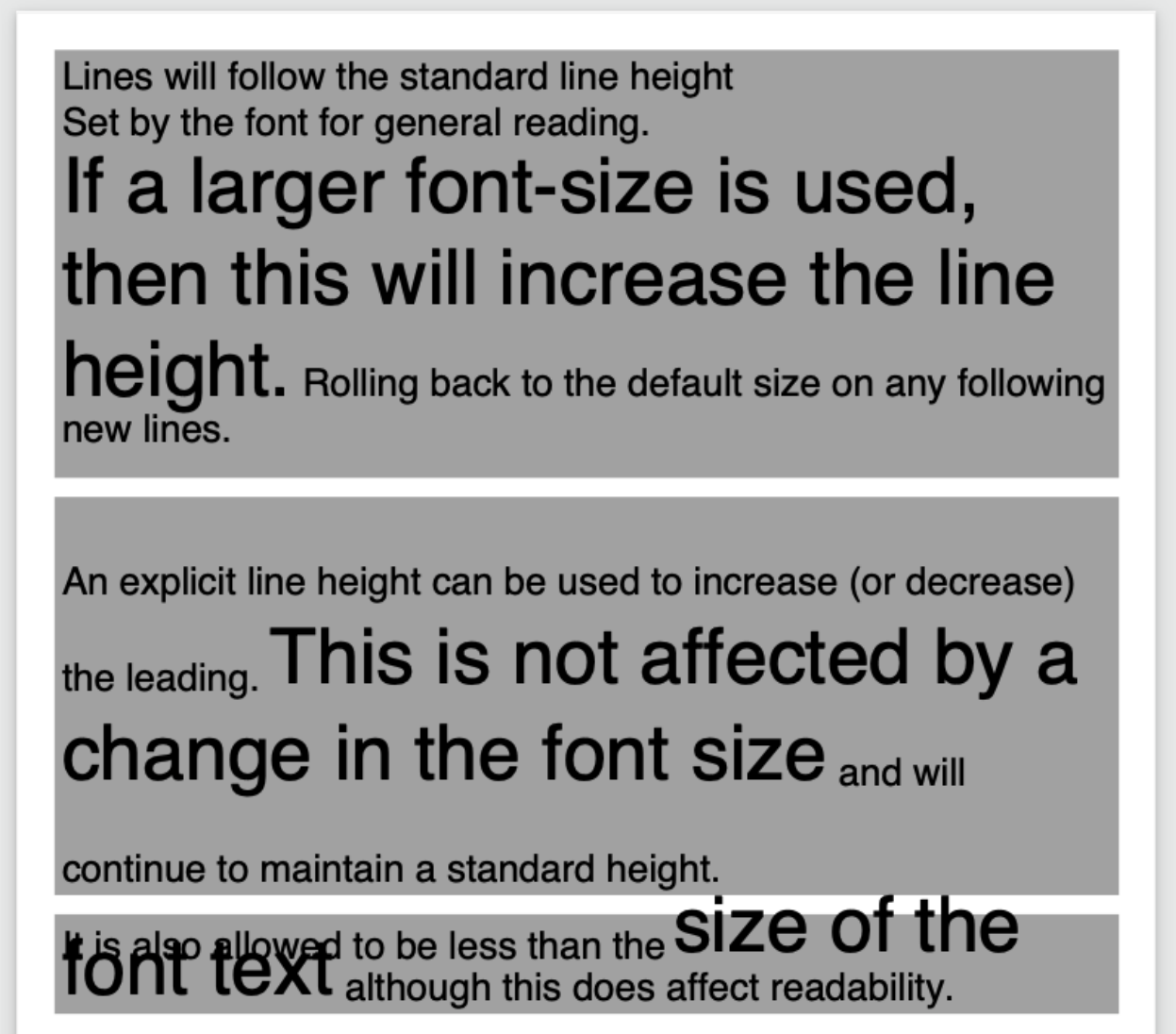
Inline components will ignore the block level style for leading. The value must be a unit value rather than a relative percent.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>Line Height</title>
    <meta name="author" content="Scryber Team" />
    <style type="text/css">

        .std-font {
            font-size: 20pt;
            background-color: #AAA;
            padding: 4pt;
            margin-bottom: 10pt;
            font-family: Geneva, Verdana, sans-serif;
        }

        .big{
            font-size: 40pt;
        }

        .high{
            line-height: 50pt;
        }

    </style>
</head>
<body style="padding: 20pt">
```

(continues on next page)

(continued from previous page)

```
    <div id="first" class="std-font">
        Lines will follow the standard line height<br/>
        Set by the font for general reading.<br/>
        <span class="big">If a larger font-size is used, then this will increase the␣
↪line height.</span>
        Rolling back to the default size on any following new lines.
    </div>

    <div class="std-font high" >An explicit line height can be used to increase (or␣
↪decrease) the
    leading. <span class="big">This is not affected by a change in the font size</
↪span> and will continue
    to maintain a standard height.</div>

    <div class="std-font" style="line-height: 22pt" >It is also allowed to be less␣
↪than
    the <span class="big">size of the font text</span> although this does affect␣
↪readability.</div>
</body>
</html>
```



10.14.7 Multi-byte Characters

Scryber supports multi-byte characters, anywhere in the document. Whether that is only a couple of characters, or whole paragraphs.

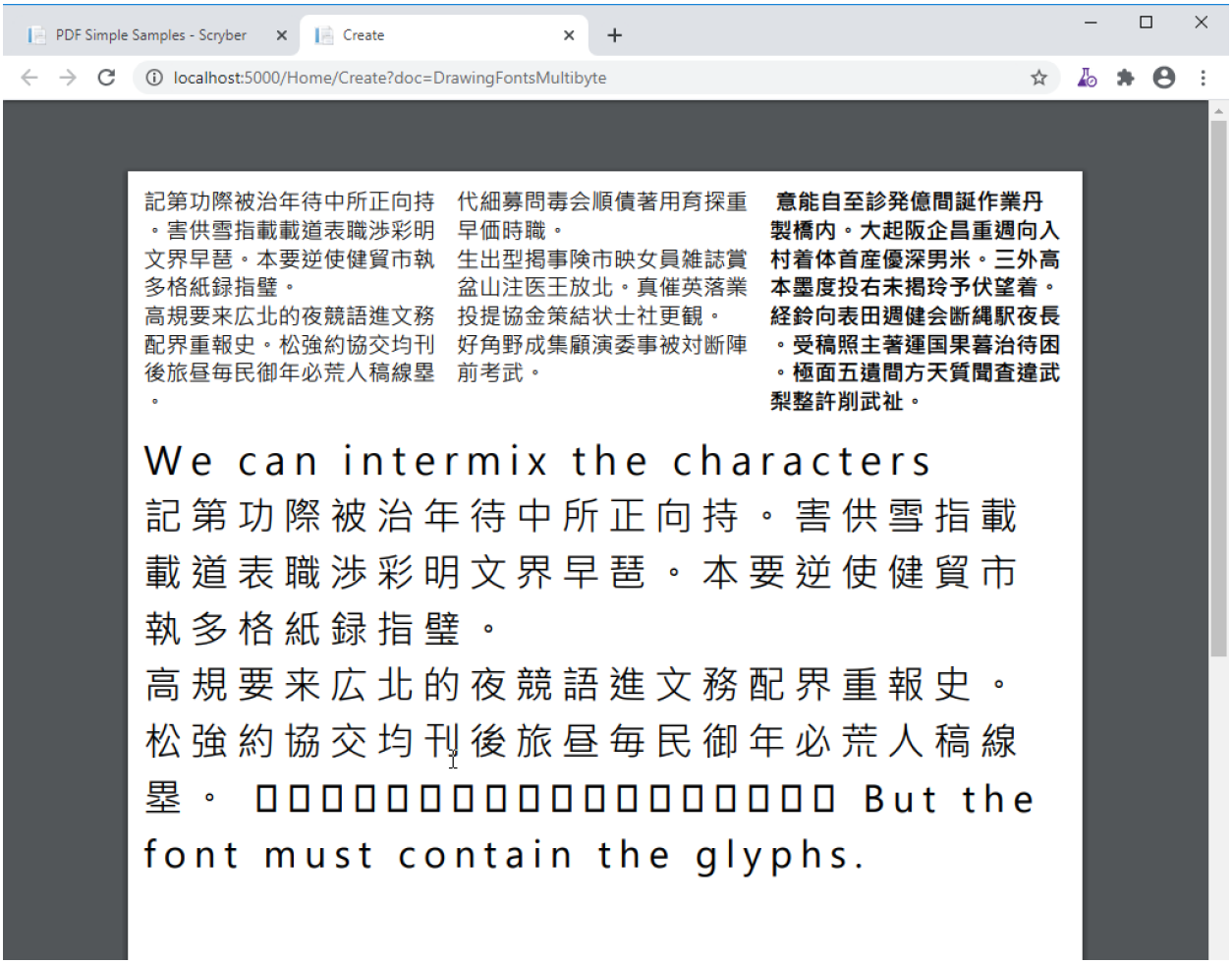
Note: The font used must also support the charcter glyphs that need to be drawn. If they are not in the font, then they cannot be rendered by the reader.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>Line Height</title>
    <meta name="author" content="Scryber Team" />
```

(continues on next page)

(continued from previous page)

```
    <style type="text/css">

        .std-font {
            font-size: 20pt;
            background-color: #AAA;
            padding: 4pt;
            margin-bottom: 10pt;
            font-family: 'Microsoft JhengHei UI', sans-serif;
        }

        .big{
            font-size: 40pt;
        }

        .high{
            line-height: 50pt;
        }

    </style>
</head>
<body style="padding: 20pt">
    <div class="std-font">




        <br />
        <br />
        <span>




        </span>
    </div>
    <!-- mixed character sets, with leading and spacing -->
    <div class="std-font">
        We can intermix the characters

        <span style='font-family:"Segoe UI";'>

            But the font must contain the glyphs.
        </span>
    </div>
</body>
</html>
```



Note: Due to the size of most unicode font files with thousands of glyphs, using and embedding a unicode font can dramatically increase the size of the pdf file. The example above came in at 23Mb without any images. Beware!

10.14.8 Right to Left

Scryber doesn't currently support Right to left (or vertical) typography. At the moment we have have not seen it done anywhere due to limitiations in postscript and the pdf document. But we will keep trying.

10.15 Textual Layout

10.15.1 Text Horizontal Alignment

Scryber supports the alignment of text at a block (rather than line level)

- Left
- Right
- Center

- Justified

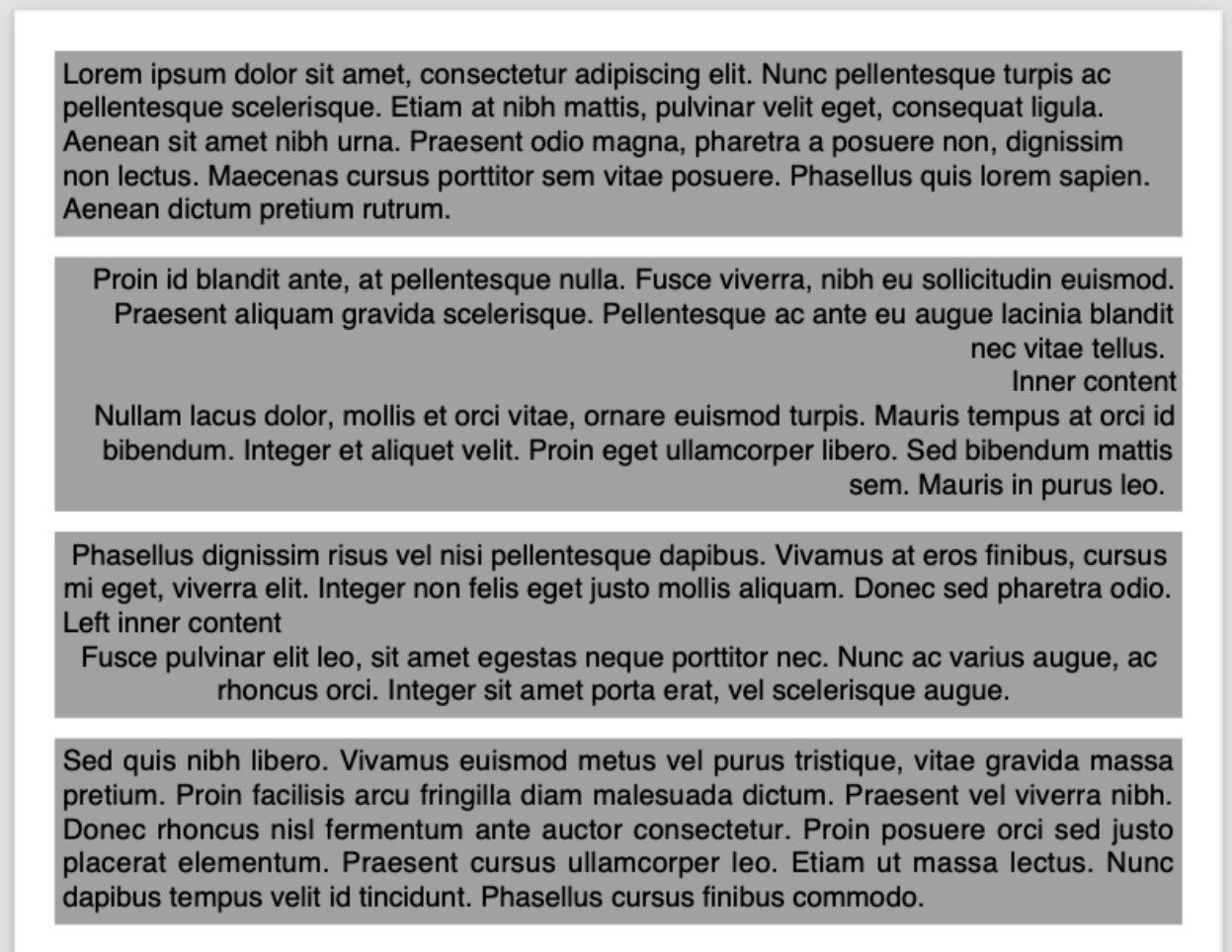
The value is inherited so that child components will be aligned in the same way, unless explicitly set differently.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>Document Text H Align</title>
    <meta name="author" content="Scryber Team" />
    <style type="text/css">

        .std-font {
            font-size: 14pt;
            background-color: #AAA;
            padding: 4pt;
            margin-bottom: 10pt;
            font-family: sans-serif;
        }

    </style>
</head>
<body style="padding: 20pt">
    <div class="std-font" style="text-align:left;">
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc pellentesque␣
↪turpis ac pellentesque scelerisque.
        Etiam at nibh mattis, pulvinar velit eget, consequat ligula.
        Aenean sit amet nibh urna. Praesent odio magna, pharetra a posuere non,␣
↪dignissim non lectus.
        Maecenas cursus porttitor sem vitae posuere. Phasellus quis lorem sapien.␣
↪Aenean dictum pretium rutrum.
    </div>

    <div class="std-font" style="text-align:right">
        Proin id blandit ante, at pellentesque nulla. Fusce viverra, nibh eu␣
↪sollicitudin euismod.
        Praesent aliquam gravida scelerisque. Pellentesque ac ante eu augue lacinia␣
↪blandit nec vitae tellus.
        <div>Inner content</div>
        Nullam lacus dolor, mollis et orci vitae, ornare euismod turpis. Mauris␣
↪tempus at orci id bibendum.
        Integer et aliquet velit. Proin eget ullamcorper libero. Sed bibendum mattis␣
↪sem. Mauris in purus leo.
    </div>

    <div class="std-font" style="text-align:center">
        Phasellus dignissim risus vel nisi pellentesque dapibus. Vivamus at eros␣
↪finibus, cursus mi eget, viverra elit.
        Integer non felis eget justo mollis aliquam. Donec sed pharetra odio.
        <div style="text-align:left">Left inner content</div>
        Fusce pulvinar elit leo, sit amet egestas neque porttitor nec.
        Nunc ac varius augue, ac rhoncus orci. Integer sit amet porta erat, vel␣
↪scelerisque augue.
    </div>

    <div class="std-font" style="text-align:justify">
```

(continues on next page)

(continued from previous page)

```
        Sed quis nibh libero. Vivamus euismod metus vel purus tristique, vitae␣
↪gravida massa pretium.
        Proin facilisis arcu fringilla diam malesuada dictum.
        Praesent vel viverra nibh. Donec rhoncus nisl fermentum ante auctor␣
↪consectetur.
        Proin posuere orci sed justo placerat elementum. Praesent cursus ullamcorper␣
↪leo.
        Etiam ut massa lectus. Nunc dapibus tempus velit id tincidunt. Phasellus␣
↪cursus finibus commodo.
    </div>
</body>
</html>
```



10.15.2 Text Vertical Alignment

The vertical alignment in text is also based on the container, and supports the following values.

- top
- middle
- bottom

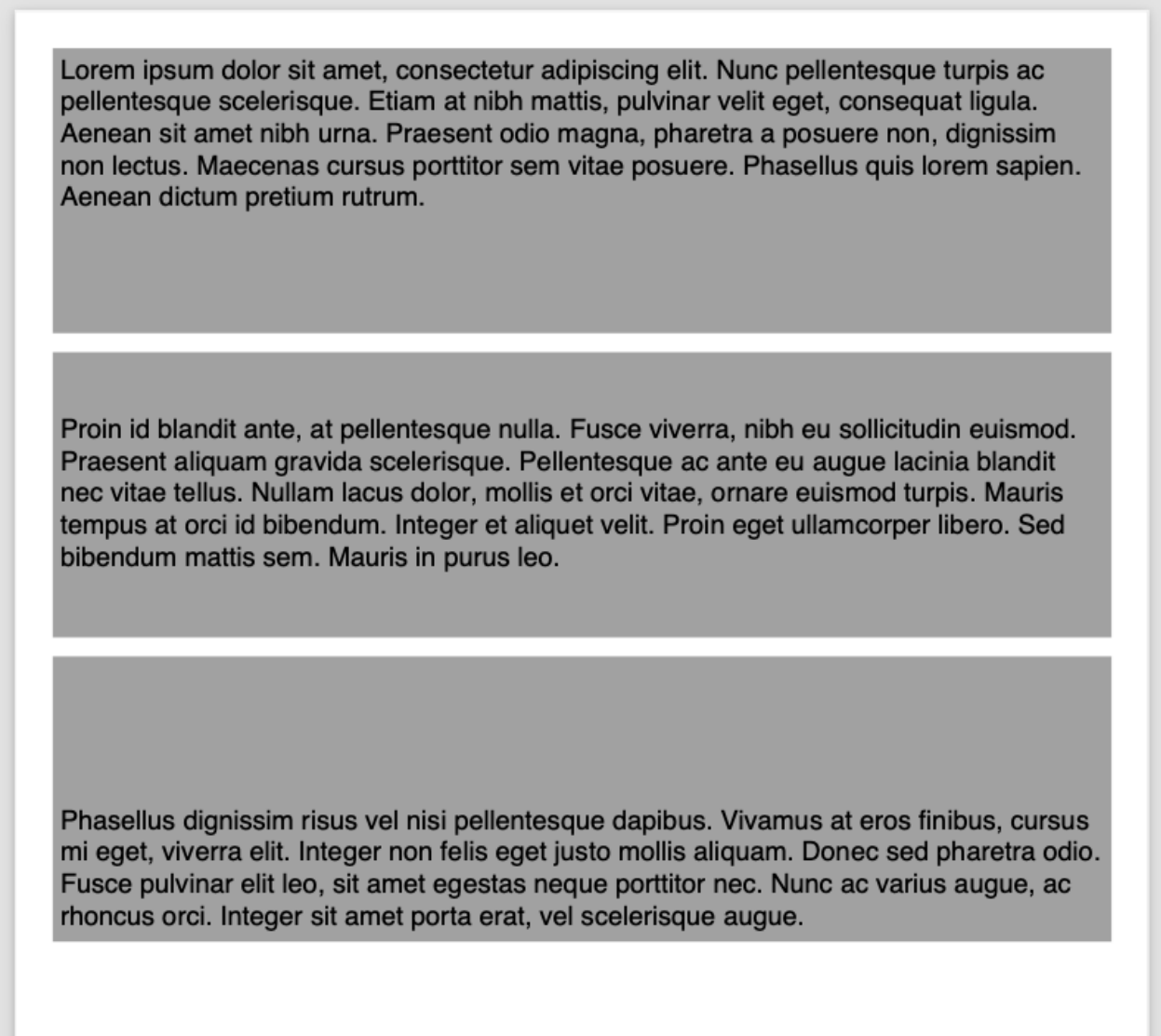
Note: The size of a block is normally shrunk to the size of the content, so vertical alignment has no visible effect. So a height must be set on the container.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>Document Text H Align</title>
    <meta name="author" content="Scryber Team" />
    <style type="text/css">

        .std-font {
            font-size: 14pt;
            background-color: #AAA;
            padding: 4pt;
            margin-bottom: 10pt;
            font-family: 'Hiragino Mincho', sans-serif;
        }

    </style>
</head>
<body style="padding: 20pt">

    <div class="std-font" style="vertical-align:top; height: 150pt">
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc pellentesque␣
↪turpis ac pellentesque scelerisque.
        Etiam at nibh mattis, pulvinar velit eget, consequat ligula.
        Aenean sit amet nibh urna. Praesent odio magna, pharetra a posuere non,␣
↪dignissim non lectus.
        Maecenas cursus porttitor sem vitae posuere. Phasellus quis lorem sapien.␣
↪Aenean dictum pretium rutrum.
    </div>

    <div class="std-font" style="vertical-align:middle; height: 150pt">
        Proin id blandit ante, at pellentesque nulla. Fusce viverra, nibh eu␣
↪sollicitudin euismod.
        Praesent aliquam gravida scelerisque. Pellentesque ac ante eu augue lacinia␣
↪blandit nec vitae tellus.
        Nullam lacus dolor, mollis et orci vitae, ornare euismod turpis. Mauris␣
↪tempus at orci id bibendum.
        Integer et aliquet velit. Proin eget ullamcorper libero. Sed bibendum mattis␣
↪sem. Mauris in purus leo.
    </div>

    <div class="std-font" style="vertical-align: bottom; height: 150pt">
        Phasellus dignissim risus vel nisi pellentesque dapibus. Vivamus at eros␣
↪finibus, cursus mi eget, viverra elit.
        Integer non felis eget justo mollis aliquam. Donec sed pharetra odio.
        Fusce pulvinar elit leo, sit amet egestas neque porttitor nec.
        Nunc ac varius augue, ac rhoncus orci. Integer sit amet porta erat, vel␣
↪scelerisque augue.
    </div>
```

(continues on next page)

(continued from previous page)

```
</body>
</html>
```



10.15.3 Differences to HTML rendering

As mentioned the vertical and horizontal alignment are declared at the container level and apply to all content within.

This is in contrast to HTML that will genrally align on the element level and flow down. It is usually easy to replicate behaviour and visual style on both.

10.15.4 Word wrapping

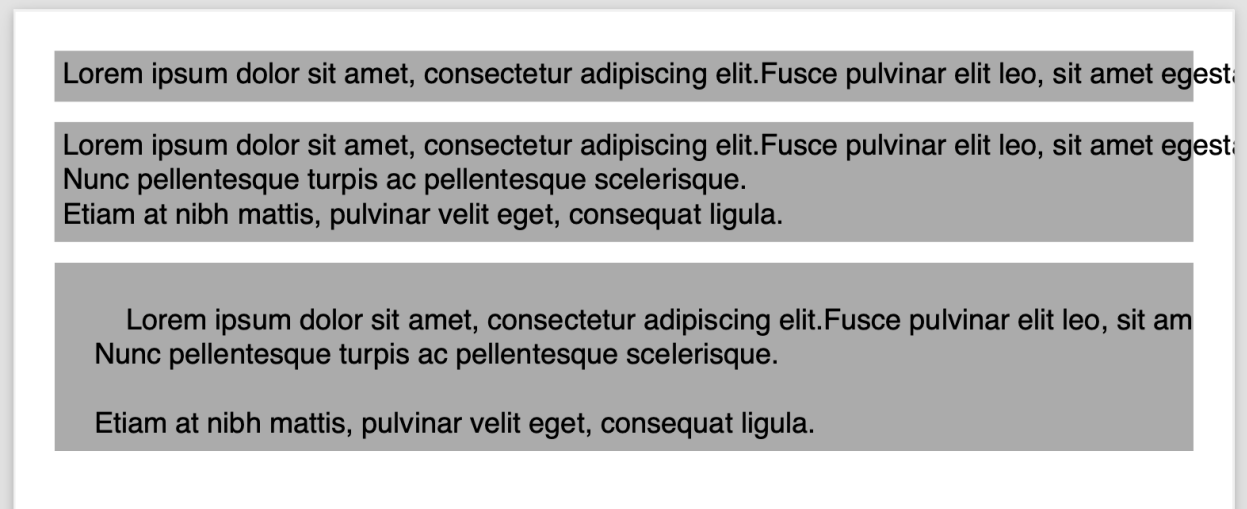
By default, scryber will wrap text around the available space and flow evenly across the page, no matter the content in the source.

If this is not the desired behaviour, then the css attributes for white-space are supported.

- nowrap - will ignore white space, AND not wrap the content when the outer edge is reached.
- pre - will take all white space into account and render content as seen.

The layout also supports the use of overflow-x and overflow-y to clip the visibility to the bounds of the container. (Scroll is not supported).

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>Document Text H Align</title>
    <meta name="author" content="Scryber Team" />
    <style type="text/css">

        .std-font {
            font-size: 14pt;
            background-color: #AAA;
            padding: 4pt;
            margin-bottom: 10pt;
            font-family: sans-serif;
        }
    </style>
</head>
<body style="padding: 20pt">
    <div class="std-font" style="white-space: nowrap">
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.Fusce␣
↪pulvinar elit leo, sit amet egestas neque porttitor nec.
        Nunc pellentesque turpis ac pellentesque scelerisque.

        Etiam at nibh mattis, pulvinar velit eget, consequat ligula.
    </div>

    <div class="std-font" style="white-space: nowrap">
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.Fusce␣
↪pulvinar elit leo, sit amet egestas neque porttitor nec.<br />
        Nunc pellentesque turpis ac pellentesque scelerisque.<br />

        Etiam at nibh mattis, pulvinar velit eget, consequat ligula.<br />
    </div>

    <div class="std-font" style="white-space:pre; overflow-x:hidden;">
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.Fusce␣
↪pulvinar elit leo, sit amet egestas neque porttitor nec.
    Nunc pellentesque turpis ac pellentesque scelerisque.

    Etiam at nibh mattis, pulvinar velit eget, consequat ligula.
    </div>

</body>
</html>
```



10.15.5 Character and Word Spacing

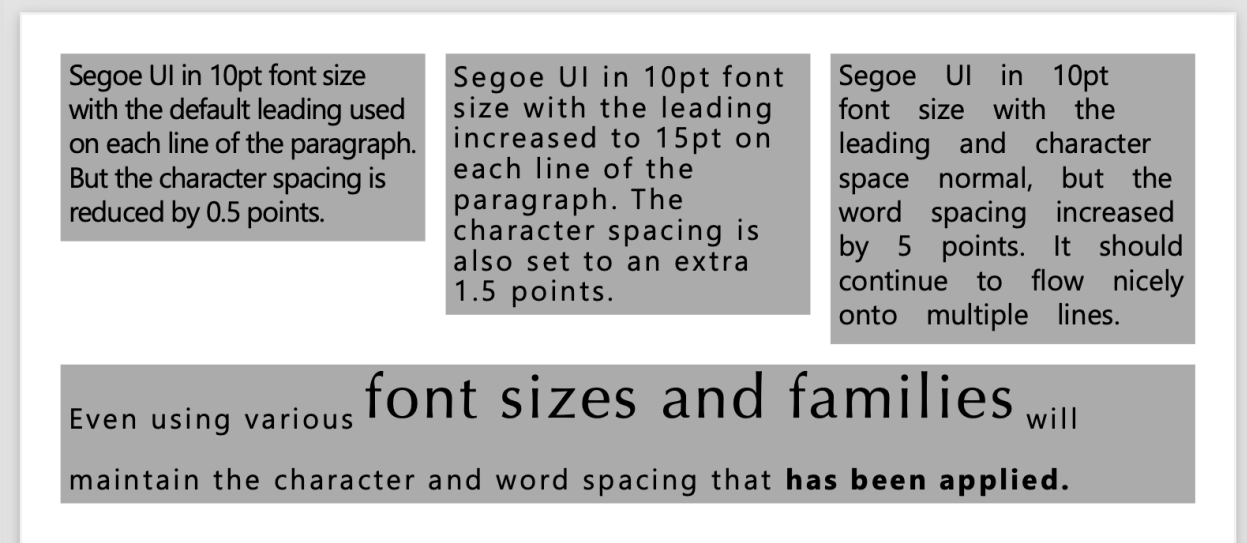
With scryber the character and word spacing is supported at the style definition level (not on the component attributes). They are less frequently used, but can help in adjusting fonts that are too narrow at a particular size, or for graphical effect.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <title>Document Character spacing</title>
    <meta name="author" content="Scryber Team" />
    <style type="text/css">

        .std-font {
            font-size: 14pt;
            background-color: #AAA;
            padding: 4pt;
            margin-bottom: 10pt;
            font-family: 'Segoe UI', sans-serif;
        }

        .narrow{ letter-spacing:-0.5pt;}

        .wide{ letter-spacing:1.5pt; line-height:15pt; }

        .wide-word{ letter-spacing: 0; word-spacing: 10pt; }

    </style>
</head>
<body style="padding: 20pt">
    <div style="column-count:3;font-size:10pt">
        <div class="std-font narrow" style="break-after:always">
            Segoe UI in 10pt font size with the default
            leading used on each line of the paragraph. But the character spacing is␣
↪reduced by 0.5 points.
```

(continues on next page)

(continued from previous page)

```
        </div>
        <div class="std-font wide" style="break-after:always">
            Segoe UI in 10pt font size with the leading increased to 15pt
            on each line of the paragraph. The character spacing is also
            set to an extra 1.5 points.
        </div>
        <div class="std-font wide-word">
            Segoe UI in 10pt font size with the leading and character space normal,␣
↪but the word
            spacing increased by 5 points. It should continue to flow nicely onto␣
↪multiple lines.
        </div>
    </div>

    <div class="std-font wide" style="line-height:30pt;" >
        Even using various
        <span style="font-size:30pt; font-family:Optima, serif;">font sizes and␣
↪families</span>
        will maintain the character and
        word spacing that <b>has been applied.</b>
    </div>

</body>
</html>
```



Note: There is a known issue with the baseline adjust on multiple font sizes that has crept in, and will hopefilly be resolved in the next release.

10.16 Colours, fills, borders and backgrounds

Colours are a standard drawing structure. They can be defined in the template and code in multiple ways.

10.16.1 Named Colours

Within a template the 140 standard HTML colours can be used These can be defined on any color style or css class,

```
<div style='background-color:red'>Red Content</div>
```

In the core library 16 pre-defined colors have been defined, and are available in the Scryber.Drawing.PDFColors class.

```
var div = new Scryber.Components.Div(){ BackgroundColor = Scryber.Drawing.PDFColors.
↪Red };
```

10.16.2 Colour Explicit Values

The colour also supports definition with explicit Hex, RGB or Grayscale values.

- **Grayscale**
  - Can be applied to an attribute with the format *G(255)*.
  - Can be applied to an attribute with the 2 character hex format *#FF*.
  - Can be assigned in code via the constructor *new PDFColor(1)*.
- **RGB color**
  - Can be applied to an attribute with the format *RGB(255,0,0)*.
  - Can be applied to an attribute with the 3 character format *#F00*
  - Can be applied to an attribute with the 6 character format *#FF0000*
  - Can be assigned in code via the constructor *new PDFColor(1,0,0)*
- **RGBA color**
  - Can only be applied to the css colors in the standard rgba(255,0,0,0.5)

```
<div style='background-color:#FF0000'>Red Content</div>
```

```
var div = new Scryber.Components.Div(){ BackgroundColor = new Scryber.Drawing.
↪PDFColor(1,0,0) };
```

```
<div style='background-color:rgba(255,0,0,0.5)'> 50% transparent red Content</div>
```

The use of transparency (opacity) is not part of the colour structure, but most drawing operations also allow a separate opacity value to be set.

Note: The internal styles have a color property and an opacity property (Background.Color, Background.Opacity) to allow image backgrounds and fills to obey this opacity.

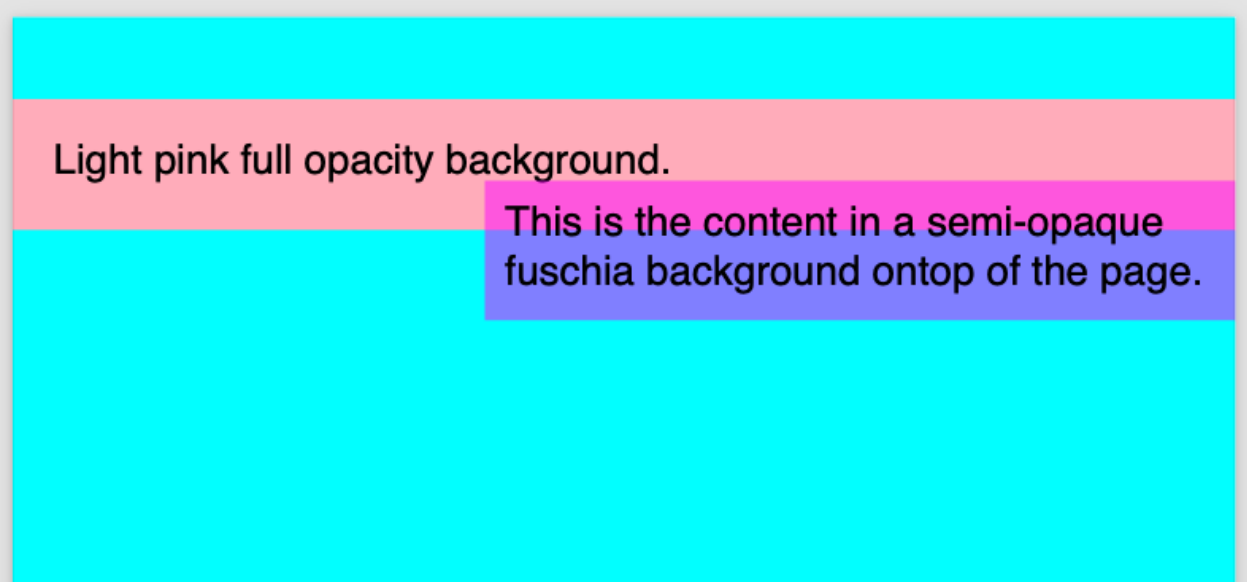
10.16.3 Using background colors

All pages and block components (see *Positioning your content*) support background colors and opacity. This can either be set via classes or explicitly on the style.

Inline components do not (currently) support rendering of a background colour.

The background is simply a solid colour from the Grayscale or RGB ranges. The background can also be specified with a fraction value from 0.0 to 1.0.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">
        body{ padding-top: 40pt;
            font-size:20pt;
            background-color: aqua;
        }
        .floating {
            position: absolute;
            top: 80pt;
            left: 230pt;
            background-color: rgba(255, 0, 255, 0.5);
            padding:10pt;
        }
    </style>
</head>
<body>
    <div style="background-color:lightpink; padding:20pt">
        Light pink full opacity background.
    </div>
    <div class="floating">
        This is the content in a semi-opaque fuschia background ontop of the page.
    </div>
</body>
</html>
```



Note: Backgrounds also support the use of single or repeating images. See *Images in documents* for details on using images backgrounds.

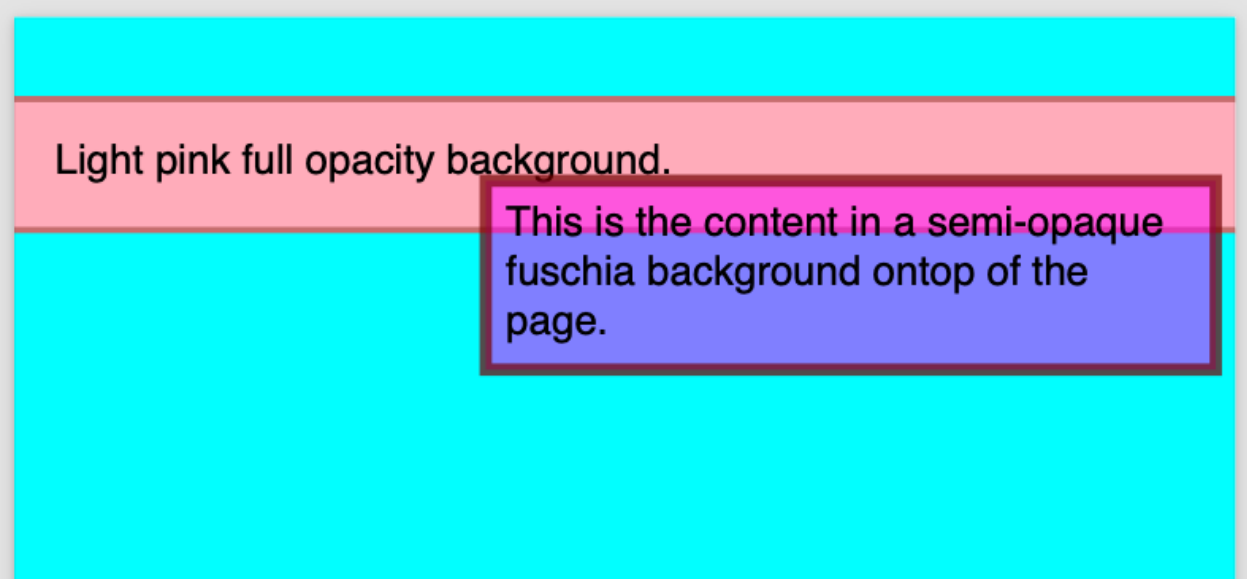
10.16.4 Using border colors

Borders apply around the edges of block components. They can be solid or dashed (see below), and have color, opaciy and width values.

Margins are outside of the border, and padding is inside. But borders do not affect either, by design.

Scryber (v5.0.5) also supports the use of border sides (border-left etc.) and corner radius.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">

        body{ padding-top: 40pt;
            font-size:20pt;
            background-color: aqua;
        }

        .floating {
            position: absolute;
            top: 80pt;
            left: 230pt;
            background-color: rgba(255, 0, 255, 0.5);
            padding:10pt;
            /* Adding the border */
            border: 6pt rgba(127,0, 0, 0.7);
            margin-right:10pt;
        }
    </style>
</head>
<body>
    <div style="background-color:lightpink; padding:20pt;
                border-top: solid 3pt #C77; border-bottom: solid #C77 3pt;">
        Light pink full opacity background.
    </div>
    <div class="floating">
        This is the content in a semi-opaque fuschia background ontop of the page.
    </div>
</body>
</html>
```



10.16.5 Using fill colors

The fill color applies to shapes and text. It is independent of background, however the same attributes apply to fills as to backgrounds.

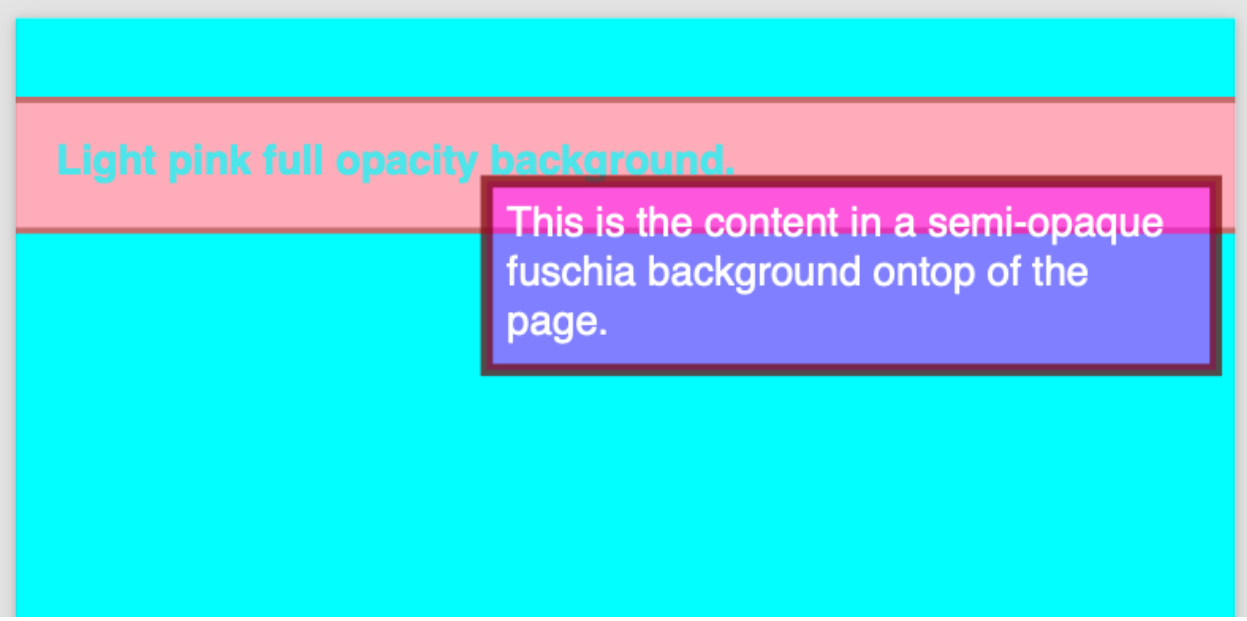
See *Drawing with SVG* for more on using fills with shapes.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">
        body{ padding-top: 40pt;
            font-size:20pt;
            background-color: aqua;
        }
        .floating {
            position: absolute;
            top: 80pt;
            left: 230pt;
            background-color: rgba(255, 0, 255, 0.5);
            padding:10pt;
            /* Adding the border */
            border: 6pt rgba(127,0, 0, 0.7);
            margin-right:10pt;

            /* Adding a fill */
            color: #FFFFFF;
        }
    </style>
</head>
<body>
    <div style="background-color:lightpink; padding:20pt;
                border-top: solid 3pt #C77; border-bottom: solid #C77 3pt;
```

(continues on next page)

(continued from previous page)

```
                color:aqua; fill-opacity: 0.7; font-weight:bold;">
        Light pink full opacity background.
    </div>
    <div class="floating">
        This is the content in a semi-opaque fuschia background ontop of the page.
    </div>
</body>
</html>
```



Note: Fills also support the use of single or repeating images. See *Images in documents* for details on using images for fills.

10.16.6 Using stroke colors

Finally stroke is around the shape or text. It supports the same properties as the border.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">
        body{ padding-top: 40pt;
            font-size:20pt;
            background-color: aqua;
        }
        .floating {
            position: absolute;
            top: 80pt;
            left: 230pt;
```

(continues on next page)

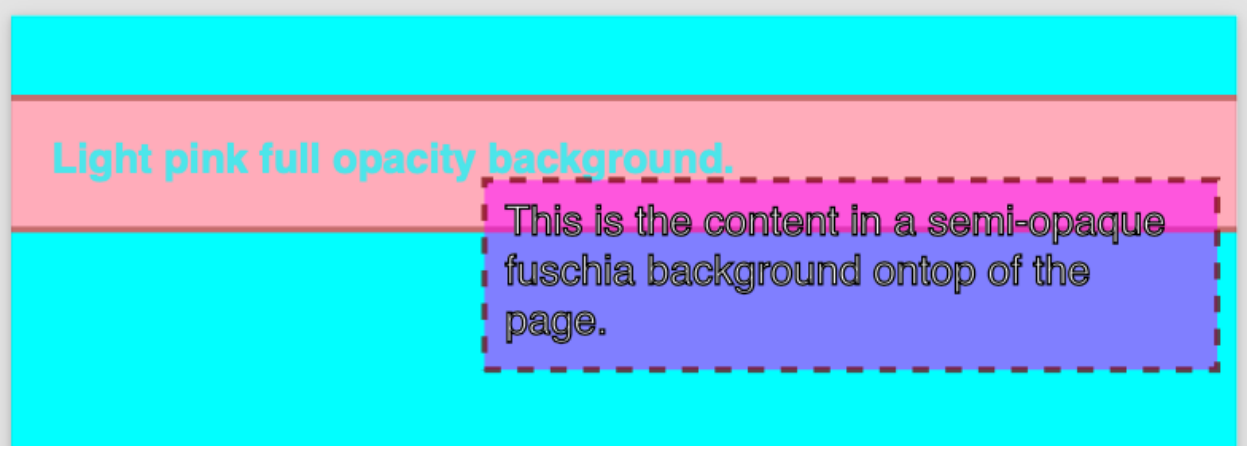
(continued from previous page)

```
            background-color: rgba(255, 0, 255, 0.5);
            padding:10pt;
            /* Adding the border */
            border: 6pt rgba(127,0, 0, 0.7);
            margin-right:10pt;

            /* Adding a fill */
            color: #FFFFFF;

            /* Adding a stroke */
            stroke: #000;
            stroke-width: 1pt;
        }
    </style>
</head>
<body>
    <div style="background-color:lightpink; padding:20pt;
                border-top: solid 3pt #C77; border-bottom: solid #C77 3pt;
                color:aqua; fill-opacity: 0.7; font-weight:bold;">
        Light pink full opacity background.
    </div>
    <div class="floating">
        This is the content in a semi-opaque fuschia background ontop of the page.
    </div>
</body>
</html>
```

10.16.7 Border and stroke styles

Both the border and stroke styles support the use of dashes. Simply changing the stroke or border style to Dashed will apply a simple dash style.

```
border: dashed 3pt rgba(127,0, 0, 0.7);
```



Note: scryber supports the solid, dashed, and dotted line styles only.

10.16.8 Binding Colors and fills

As with all things scryber. The styles are all bindable to parameters and data, so regular colours could be defined and then used in places throughout the styles and components.

See *Dynamic content in your template* for an example.

10.17 Units and measures

Within scryber all drawing and positioning is based from the top left of the page. Scryber allows the definition of a dimension based on a number of positioning and sizing structures. All based around the **Unit** of measure.

In all the examples so far we have used pt (points) as the unit of measure, but scryber also supports the use of millimeters (mm), inches (in) and also pixels (px) as postfix units. A pixel is translated to 1/96 th of an inch for printing.

Note: Scryber does not support the use of relative dimensions: em, rem, vh etc. The only exception is the use of 100% on widths.

- **PDFUnit**
  - This is the base single dimension value.
  - Its default scale is the prinding standard points unit (1/72nd of an inch).
  - Values can also be specified in millimeters (mm) and inches (in) as well as explicitly in points.
  - Units are used in many places in xml templates
  - e.g. 72, 72pt, 1in, 25.4mm would all represent a 1.0 inch dimension.
  - Units can directly be cast and converted from integer and double values, or constructed in code.
  - see reference/drawing_unit for more information.
- **PDFSize**

  - This is a width and height dimension with 2 PDFUnits.
  - Units can be mixed and matched within a size, but are generally only used internally for calculation
  - e.g. *72pt 1in* is a 1.0 inch wide and high

- **PDFPoint**
  - This represents a location on a page or container with an x and y component.
  - Again units can be mixed and matched within a point, but are generally only used internally for calculation
  - e.g. *72pt 1in* is 1 inch in from the left of the container and 1 inch down.

- **PDFThickness**
  - A thickness represent 4 dimensions around a square.
  - It follows the same order as html starting at the top and moving in a clockwise direction to the right, bottom and left.
  - It can be defined with 1, 2 or 4 values as a string, where 1 dimension refers to all values, 2 is the vertical and then horizontal and all 4 are explicit.
  - Thicknesses are used by the margins, padding, clipping attributes on components.
  - e.g. *25.4mm*, *1in 72pt* or *72 72 72 72* are all equivalent to 1 inch thickness all around.
  - see reference/drawing_thickness for more information.

- **PDFRect**
  - A rectagle is represented by 4 dimensions forming the x, y, width and height of a rectangle.
  - They can be mixed and matched in units, but are generally only used internally for calculation.
  - This should not be confused with the PDFRectangle used in drawing (see *Drawing with SVG*)
  - see reference/drawing_rect for more information.

10.17.1 Units use in templates

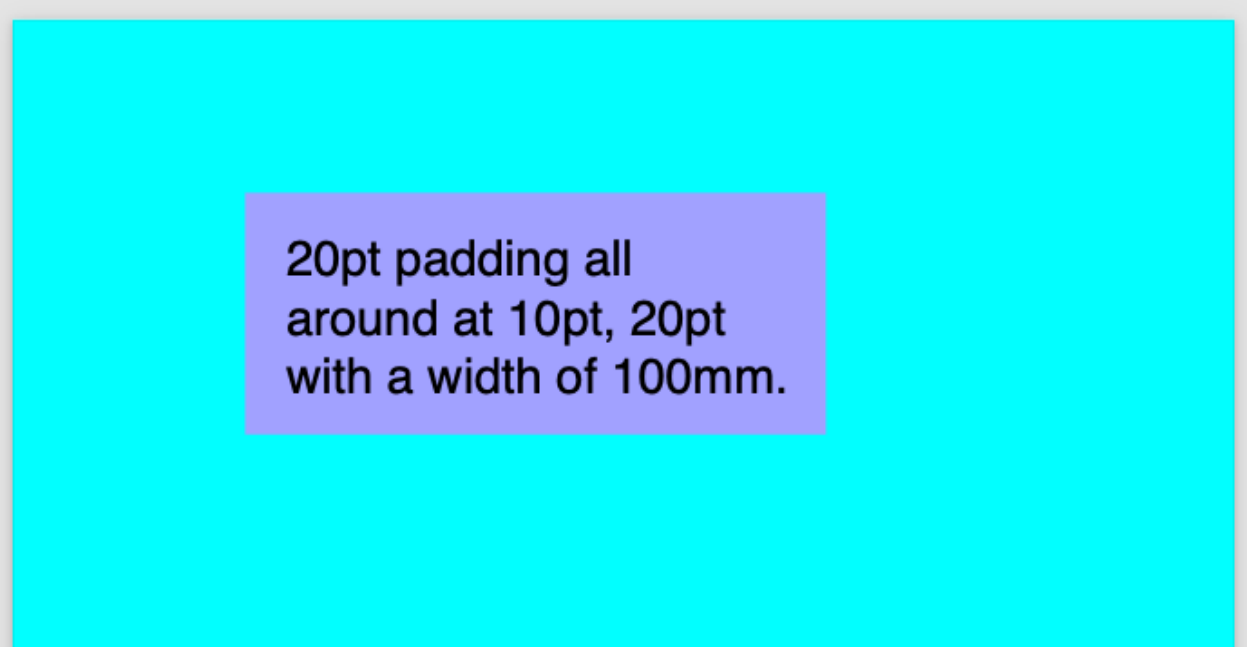
When using units in xml templates its easy just to provide the values. For example the following will add an absolutely positioned Div on a page with some thickness padding textual content

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">
        body{
            background-color: aqua;
        }

        .positioned {
            position: absolute;
            top: 30mm;
            left: 40mm;
            width: 100mm;
            padding:20pt 0.25in 4mm 20pt;
```

(continues on next page)

(continued from previous page)

```
            background-color: #AAAAFF;
        }

    </style>
</head>
<body>
    <div class="positioned">
        20pt padding all around at 10pt, 20pt with a width of 100mm.
    </div>
</body>
</html>
```



10.17.2 Units in code

The same could have be achieved in code using the Unit and Thickness constructors.

All the dimensions have a range of constructors, casting and parsing options as needed.

```
//using Scryber.Drawing

PDFUnit unit1 = 20; //implicit cast to 20pts
var unit2 = (PDFUnit)72; //explicit cast to 72 points (1 inch)
var unit3 = new PDFUnit(1, PageUnits.Inches); //explicit unit scale

var pt1 = new PDFPoint(20,72); //defaults to points
var pt2 = new PDFPoint(unit1, unit2); //explicit unit dimensions

var thick1 = new PDFThickness(unit3); //Applies to all with a PDFUnit value
var thick2 = new PDFThickness(10,20,10,20); //Applies explicit values to each␣
↪dimensions

var rect = PDFRect.Empty; //Set to Zeroed values.
rect.Inflate(thick2); //Then inflate the rectangle by the thickness.
```

(continues on next page)

(continued from previous page)

```
var rect2 = PDFRect.Parse("12pr 10pt 100pt 2in"); //And all support parsing too.
```

10.17.3 One Hundred Percent

The special value of 100% for width applies true to the underlying FullWidth style value.

By default div's have a FullWidth of true, so they will be 100% wide, but tables, lists etc do not. By specifying a width of 100% on these, they will use all the available space.

See *Positioning your content* for more information.

10.17.4 Overiding relative units

Finally: If there is an existing template or file being used, then overriding any relative styles can be done using the @media print rule - so it will only be used by scryber (or when the document is printed anyway).

10.18 Images in documents

Scryber supports most standard image types, including the following.

- GIF - 1 & 8 bit,
- TIFF - 1, 8, 24 and 32 bit alpha, Indexed, RGB and CMYK.
- PNG - 8, 24 and 32 bit alpha, RGB.
- JPEG - 24 bit RGB.

Adding an image to the output is as simple as putting an ** in the template. If the image has an alpha channel this will automatically be applied to the image.

10.18.1 Image Source Paths.

The source for an image path can be referenced relative to the current file being parsed.

e.g. ../../Content/images/Toroid32.tiff

Or it can be the full url or full file path to the image.

e.g. http://localhost:5000/images/Toroid32.tiff

Note: If the image component is declared on a file referenced from the document, then the image path should still be relative to the file where the image component is declared. Not the top level document.

10.18.2 Explicit image data

Data can be set on the image explicitly when loaded, using the Scryber.Drawing.PDFImageData class. This class has a range of static methods to load the images from existing bitmaps, local files or streams.

And the image data can then be assigned to an image component directly.

```
using (var stream = GetMyImageData())
{
    var img = doc.FindAComponentById("myImage") as Image;
    if (null != img)
        img.Data = PDFImageData.LoadImageFromStream("IdentifyingKeyForImage",␣
↪mybitmapstream);
}

//Other overloads

//PDFImageData.LoadImageFromBitmap();
//PDFImageData.LoadImageFromLocalFile();
//PDFImageData.LoadImageFromURI();
```

10.18.3 Binding Image data

Binding to parameter data is also supported in the img tag using the data-img attribute.

```
<img alt='Data bound image' data-img='{@:model.imagedata}' >
```

And this can be loaded and set in the usual way.

```
using (var stream = GetMyImageData())
{
    var model = new {
        imageData = PDFImageData.LoadImageFromStream("IdentifyingKeyForImage",␣
↪mybitmapstream)
    };
    doc.Params["model"] = model;
}
```

10.18.4 Image Sizing

Without an explicit size the images will be rendered at the natural size, if it fits in the container.

If it does not fit in the container it will be reduced in size proportionally to fit. Setting a height or width will also constrain the image proportionally. If both are set then the image will use these and it may no longer be proportional.

See *Sizing your content* and *Column layout* for more about sizing images with widths and heights.

10.18.5 Pulling it together

Once the path or data are set, it's just a case of rendering the document in the normal way.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">

        img.toroid{
```

(continues on next page)

(continued from previous page)

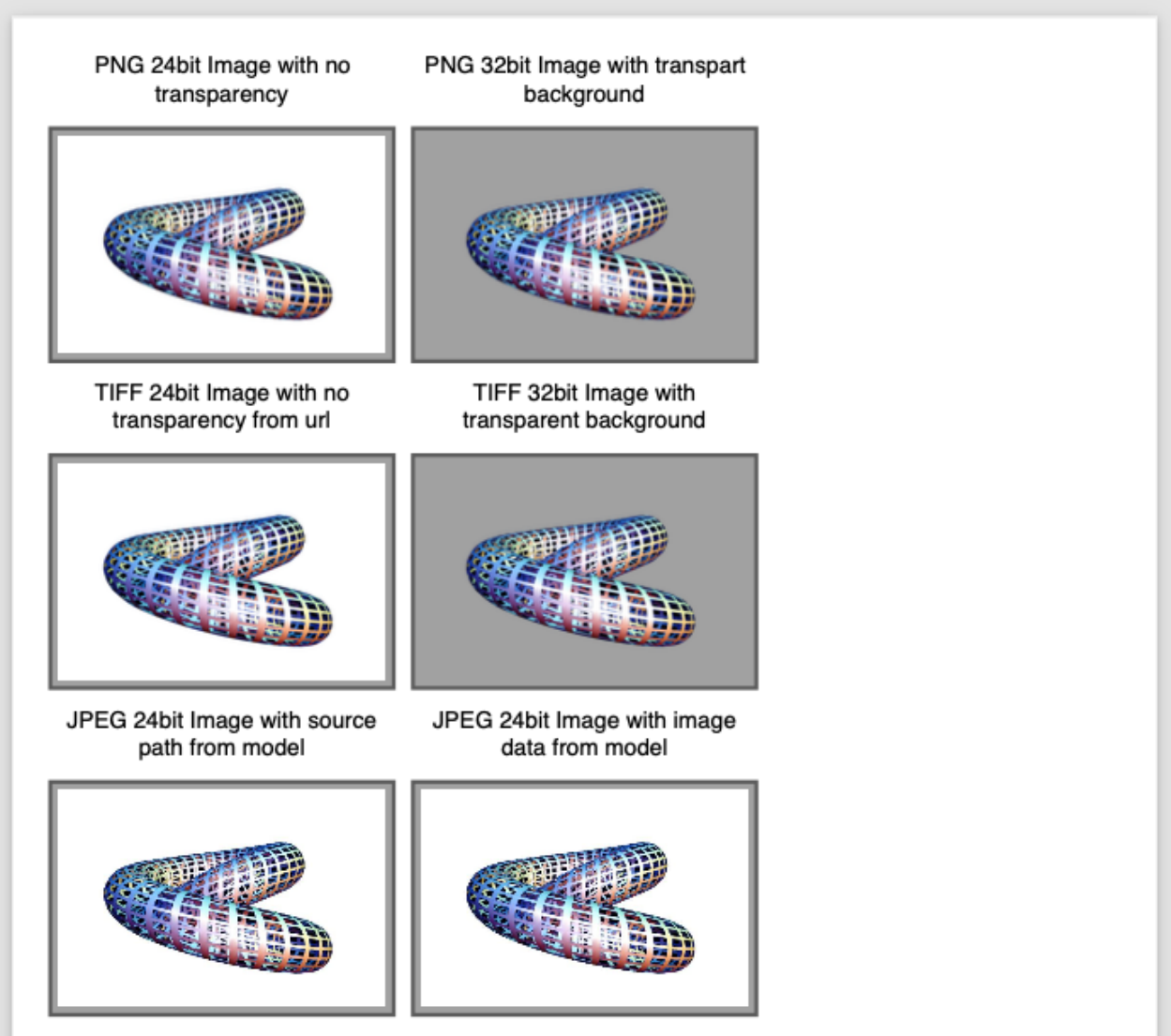
```
            border: solid 2pt #666;
            background-color:#AAA;
            padding: 4pt;
            margin: 10pt 0;
            break-after:always;
        }

        div.wrap {
            column-count:3;
            font-size:12pt;
            text-align:center;
        }

    </style>
</head>
<body style="padding:20pt;">
    <div class="wrap">
        <span>PNG 24bit Image with no transparency</span>
        <img class="toroid" src="./images/Toroid24.png" alt="24 bit PNG file" />

        <span>PNG 32bit Image with transpart background</span>
        <img class="toroid" src="./images/Toroid32.png" alt="32 bit PNG file" />
    </div>

    <div class="wrap">
        <span>TIFF 24bit Image with no transparency from url</span>
        <img class="toroid"
            src="https://raw.githubusercontent.com/richard-scryber/scryber.core/
↪master/docs/images/Toroid24.tiff"
            alt="24 bit Tiff file" />

        <span>TIFF 32bit Image with transparent background</span>
        <img id="tiff32" class="toroid" alt="32 bit TIFF file" />
    </div>

    <div class="wrap">
        <span>JPEG 24bit Image with source path from model</span>
        <img class="toroid" src="{@:model.jpgSrc}" alt="24 bit JPEG file" />

        <span>JPEG 24bit Image with image data from model</span>
        <img class="toroid" data-img="{@:model.jpgData}" alt="32 bit JPEG file" />
    </div>

</body>
</html>
```

```
var path = System.Environment.CurrentDirectory;
var docPath = System.IO.Path.Combine(path, "../../../Content/HTML/documentation.html
↪");

using (var doc = Document.ParseDocument(docPath))
{
    //pass paramters as needed, supporting simple values, arrays or complex classes.
    var img = doc.FindAComponentById("tiff32") as Image;

    if(null != img)
```

(continues on next page)

(continued from previous page)

```
        img.Source = System.IO.Path.Combine(path, "../../../Content/HTML/Images/
↪Toroid32.tiff");

    var jpgSrc = System.IO.Path.Combine(path, "../../../Content/HTML/Images/Toroid24.
↪jpg");
    var jpgData = PDFImageData.LoadImageFromLocalFile(jpgSrc);

    var model = new
    {
        jpgSrc = jpgSrc,
        jpgData = jpgData
    };

    doc.Params["model"] = model;

    using (var stream = DocStreams.GetOutputStream("documentation.pdf"))
    {
        doc.SaveAsPDF(stream);
    }
}
```

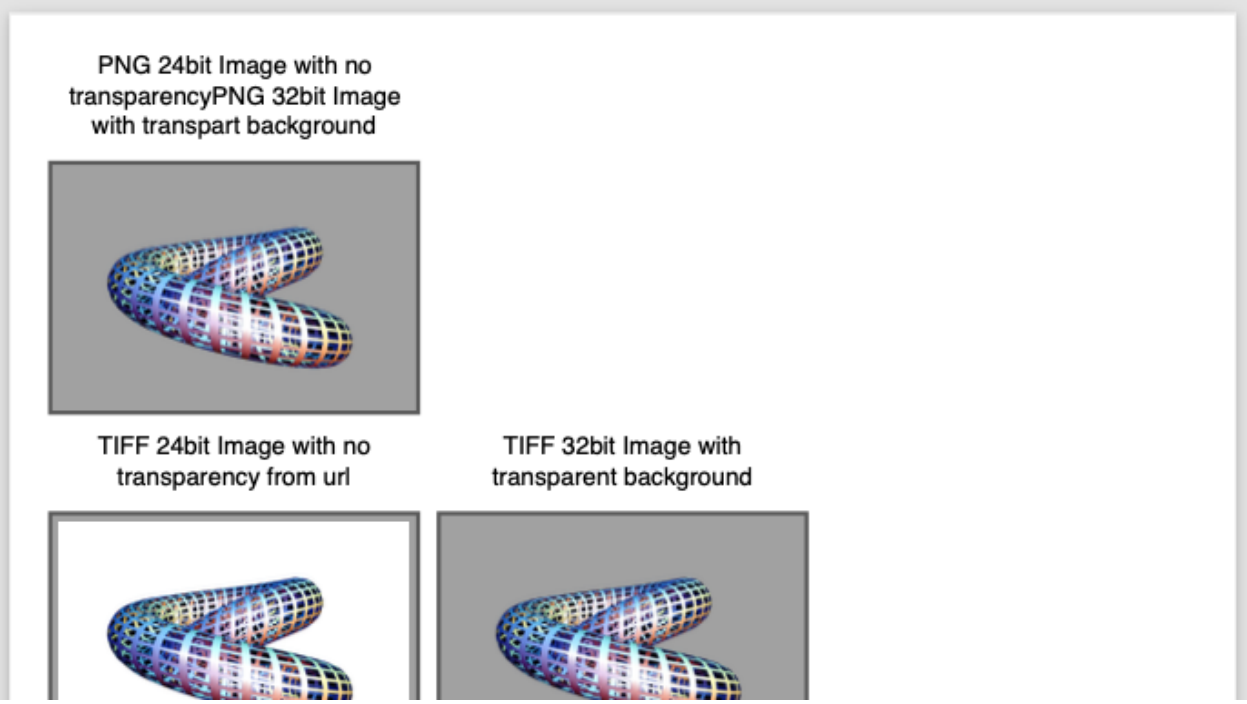


Note: Pre v5.0.5 of the package, the imgate data attribute was img-data. This will continue to work, but is considered obselete and the preferred attribute is data-img.

10.18.6 Not found Images

If a path is set on an image but not resolved then scryber will simply treat the flow as normal.

```
<div class="wrap">
    <span>Replacing with a non-existant image.</span>
    <img class="toroid" src="DoesnotExist.png" alt="24 bit PNG file" />

    <span>PNG 32bit Image with transpart background</span>
    <img class="toroid" src="./images/Toroid32.png" alt="32 bit PNG file" />
</div>
```



There will however, if we switch them on, be an error in the **always useful** logs

```
<?xml version="1.0" encoding="utf-8" ?>
<?scryber append-log='true' ?>
```

Document Log

Time (ms)	Level	Category	Message
236.9234	Message	Document	Beginning Document Initialize
240.8789	Message	Document	Completed Document Initialize
241.0339	Message	Document	Beginning Document Load
260.2448	Message	Document	Completed Document Load
260.8038	Message	Document	Beginning Document Databind
265.3974	Message	meta	Updating the document information and restrictions
269.4228	Message	Document	Completed Document Databind
280.4843	Message	meta	Updating the document information and restrictions
281.6611	Message	Document	Beginning Document layout
292.7626	Message	Document Layout	Starting the layout of page sect1
309.1804	Message	Layout Engine	Laying out the page component 'sect1' at page index 0

Time (ms)	Level	Category	Message
534.6422	Error	Document	Could not load the image data for 'DoesnotExist.png'. Failed with message : The file at the path '/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/Content/HTML/DoesnotExist.png' does not exist. (Parameter 'path') System.ArgumentNullException: The file at the path '/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/Content/HTML/DoesnotExist.png' does not exist. (Parameter 'path') at Scryber.Drawing.PDFImageData.LoadImageFromLocalFile(String path, IPDFComponent owner) in /Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Drawing/Drawing/PDFImageData.cs:line 585 at Scryber.Components.Document.LoadImageData(IPDFComponent owner, String path) in /Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Components/Components/Document.cs:line 2237
534.8559	Error	Document	Could not load the image data for 'DoesnotExist.png'. Failed with message : The file at the path '/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/Content/HTML/DoesnotExist.png' does not exist. (Parameter 'path') System.ArgumentNullException: The file at the path '/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/Content/HTML/DoesnotExist.png' does not exist. (Parameter 'path') at Scryber.Drawing.PDFImageData.LoadImageFromLocalFile(String path, IPDFComponent owner) in /Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Drawing/Drawing/PDFImageData.cs:line 585 at Scryber.Components.Document.LoadImageData(IPDFComponent owner, String path) in /Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Components/Components/Document.cs:line 2237
535.1260	Error	Image	Missing Image replaced: The image path could not be resolved, or the image could not be loaded from path 'DoesnotExist.png'

10.18.7 Enforcing not found exceptions

This is the safest behaviour, but if this is not the desired behaviour, then there are a couple of options that change the behaviour to allow missing images.

- On the image itself, if you know it may not be found, simply set the attribute data-allow-missing-images to false.
- On the document as a whole with the AllowMissingImages property.
- Change the behaviour of scryber as a whole to disallow missing images using the configuration options. See scryber_configuration

```
<img class="toroid" data-allow-missing-images="false" src="DoesnotExist.png" alt="24␣
↪bit PNG file" />
```

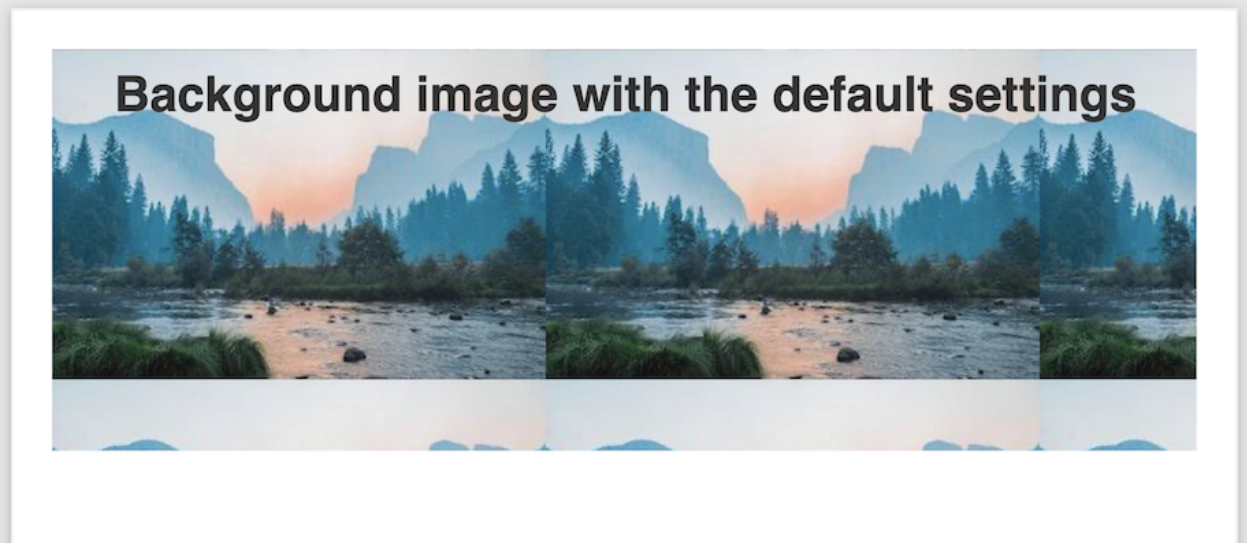
```
doc.RenderOptions.AllowMissingImages = false;
```

If either are set to false an exception will be raised (i.e. it is not possible to disallow images at the document level, but allow on some images).

10.19 Images as backgrounds and fills

Images are also supported on the backgrounds of block level components (see *Positioning your content*), and of fills for shapes, text, etc.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">

        div.bg {
            background-image: url("./images/landscape.jpg");
            min-height: 260px;
            text-align:center;
            color:white;
            font-family: sans-serif;
            font-size:larger;
            font-weight:bold;
            padding-top:10pt;
        }

    </style>
</head>
<body style="padding:20pt;">
    <div class="bg" style="">
        <span>Background image with the default settings</span>
    </div>

</body>
</html>
```



The background has been drawn with the image repeating from the top left corner at its natural size (or default 96ppi), clipped to the boundary of the container.

Along with specifying the image background, there are various other options for how the pattern is laid out that will change the defaults of how the image repeats.

10.19.1 Background Size

The background size option can either be a specific size, or 'cover' which will cover the entire container as a single image.

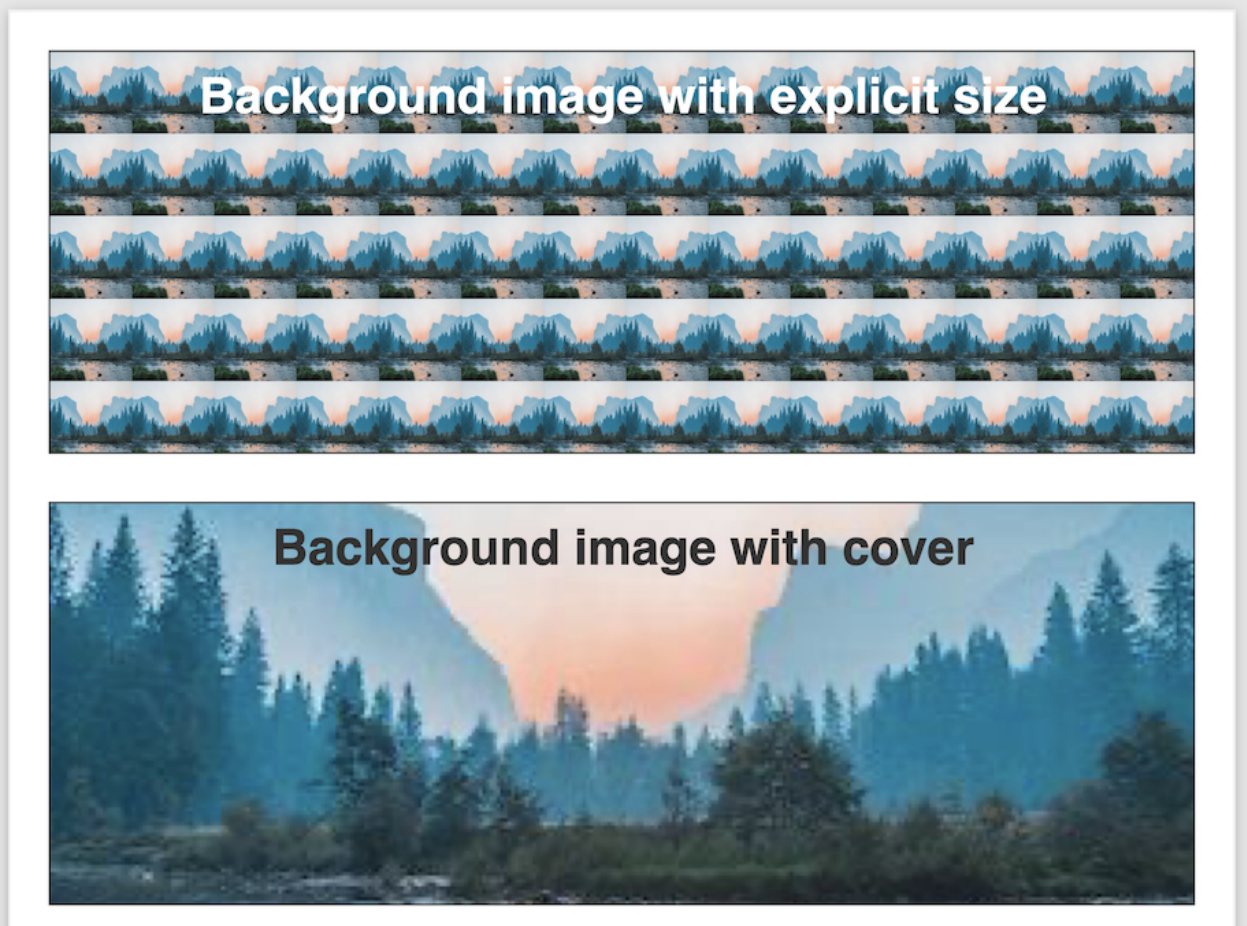
(Scryber does not currently support 'contain' but it's on our roadmap).

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">

        div.bg {
            background-image: url("./images/landscape.jpg");
            min-height: 260px;
            text-align:center;
            color:#333;
            font-family: sans-serif;
            font-size:larger;
            font-weight:bold;
            padding-top:10pt;
            border:solid 1px #333;
        }

    </style>
</head>
<body style="padding:20pt;">

    <div class="bg" style="background-size: 40pt 40pt; color:white;">
```

(continues on next page)

(continued from previous page)

```
        <span>Background image with explicit size</span>
    </div>
    <br/>
    <div class="bg" style="background-size:cover">
        <span>Background image with cover</span>
    </div>

</body>
</html>
```



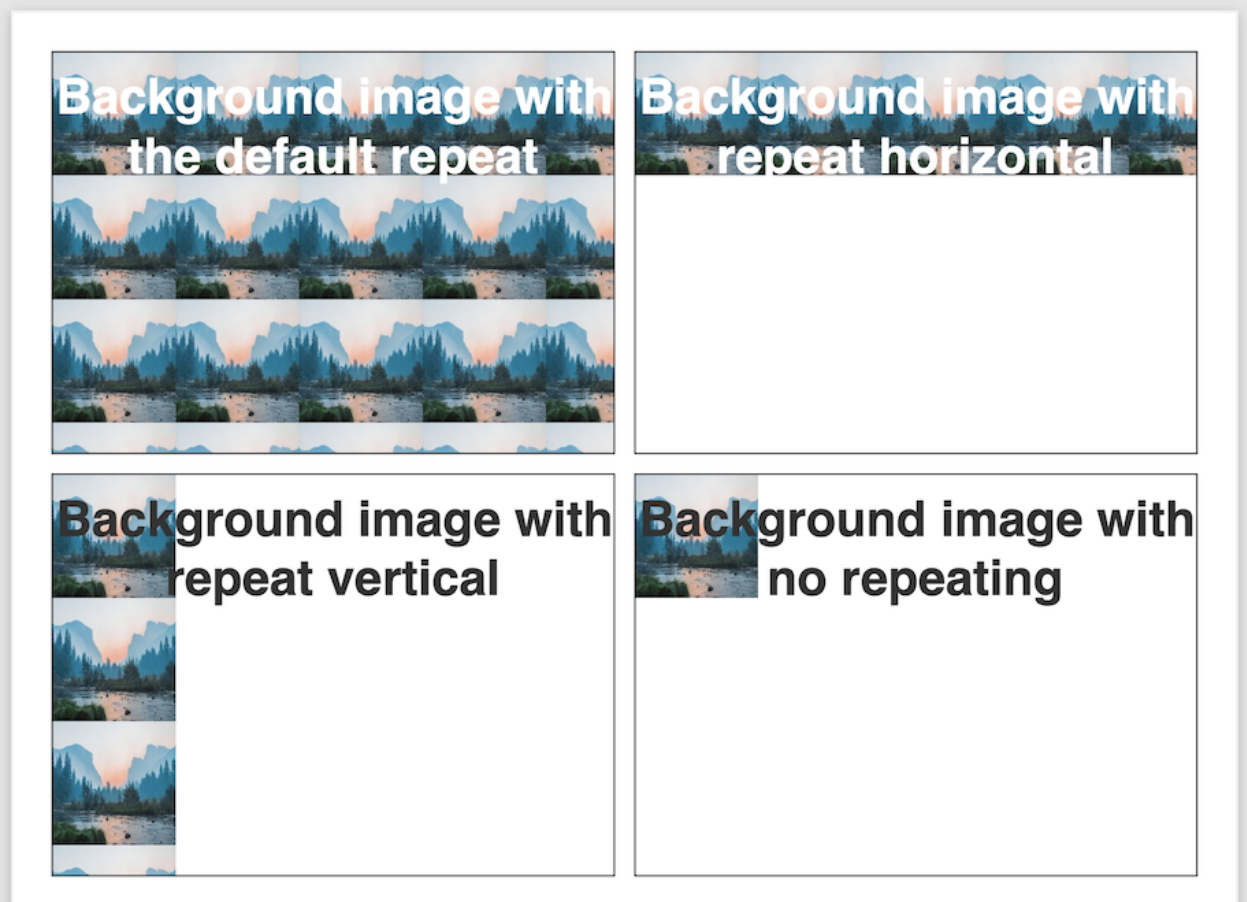
10.19.2 Background Repeat

The options for the background repeating are:

- repeat - The default value, where the image repeats both X and Y directions.
- repeat-x - The background will only repeat in the X (horizontal) direction.
- repeat-y - The background will only repeat in the Y (vertical) direction.
- none - The background will only be shown once.

These can be applied with a size, but will not affect anything if the size is cover.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">

        div.bg {
            background-image: url("./images/landscape.jpg");
            min-height: 260px;
            text-align:center;
            font-family: sans-serif;
            font-size:larger;
            font-weight:bold;
            padding-top:10pt;
            border:solid 1px #333;
            /* consistent size across all */
            background-size: 60pt 60pt;
        }

    </style>
</head>
<body style="padding:20pt;">


    <div style="column-count:2; margin-bottom: 10pt; color:white;">
        <div class="bg" style="background-repeat:repeat; break-after:always;">
            <span>Background image with the default repeat</span>
        </div>
        <div class="bg" style="background-repeat:repeat-x">
            <span>Background image with repeat horizontal</span>
        </div>
    </div>

    <div style="column-count:2; color:#333;">
        <div class="bg" style="background-repeat:repeat-y; break-after:always;">
            <span>Background image with repeat vertical</span>
        </div>

        <div class="bg" style="background-repeat:no-repeat">
            <span>Background image with no repeating</span>
        </div>

    </div>
</body>
</html>
```



10.19.3 Background Position

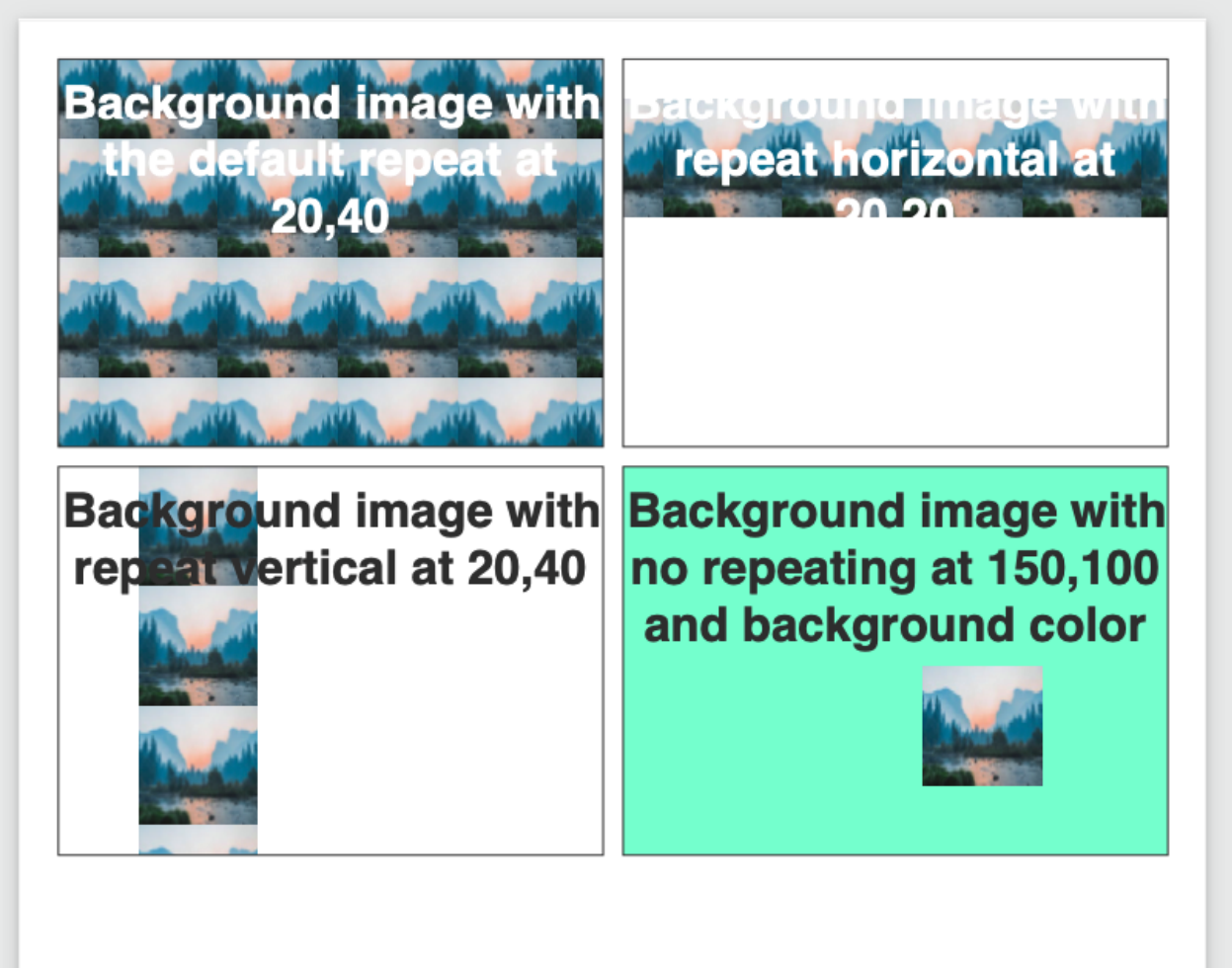
- **The starting position of the pattern.**
  - x-pos - Determines the horizontal offset of the rendered background image in units.
  - y-pos - Determines the vertical offset of the rendered background image in units.
- **The pattern repeat step.**
  - x-step - Sets the horizontal offset between repeating patterns, which can be more or less than the size of the rendered image.
  - y-step - Sets the vertical offset between repeating patterns, which can be more or less than the size of the rendered image.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <style type="text/css">

        div.bg {
            background-image: url("./images/landscape.jpg");
```

(continues on next page)

(continued from previous page)

```
            min-height: 260px;
            text-align: center;
            font-family: sans-serif;
            font-size: larger;
            font-weight: bold;
            padding-top: 10pt;
            border: solid 1px #333;
            /* consistent size across all */
            background-size: 60pt 60pt;
        }
    </style>
</head>
<body style="padding:20pt;">


    <div style="column-count:2; margin-bottom: 10pt; color:white;">
        <!-- Position value for x and y -->
        <div class="bg"
            style="background-repeat:repeat;
                    background-position: 20pt 20pt;
                    break-after:always;">
            <span>Background image with the default repeat at 20,20</span>
        </div>
        <!-- Single value should be applied to both x and y -->
        <div class="bg"
            style="background-repeat:repeat-x;
                    background-position: 20pt">
            <span>Background image with repeat horizontal at 20,20</span>
        </div>
    </div>

    <div style="column-count:2; color:#333;">
        <!-- x and y as individual properties -->
        <div class="bg"
            style="background-repeat: repeat-y;
                    background-position-x: 20pt;
                    background-position-y: 40pt;
                    break-after: always;">
            <span>Background image with repeat vertical at 20,20</span>
        </div>
        <!-- Single repeat with a bakground color -->
        <div class="bg"
            style="background-repeat: no-repeat;
                    background-position: 150pt 100pt;
                    background-color: aquamarine">
            <span>Background image with no repeating at 150,100 and background color</
↪span>
        </div>

    </div>
</body>
</html>
```



10.19.4 Images as fills

Scryber also supports images as fills. See the SVG documentation for this.

10.20 Drawing with SVG

Scryber includes the drawing capability with a subset of SVG capabilities.

- Lines
- Recangles
- Elipses
- Polygons
- Bezier Curves
- Groups
- Text
- Use and definitions

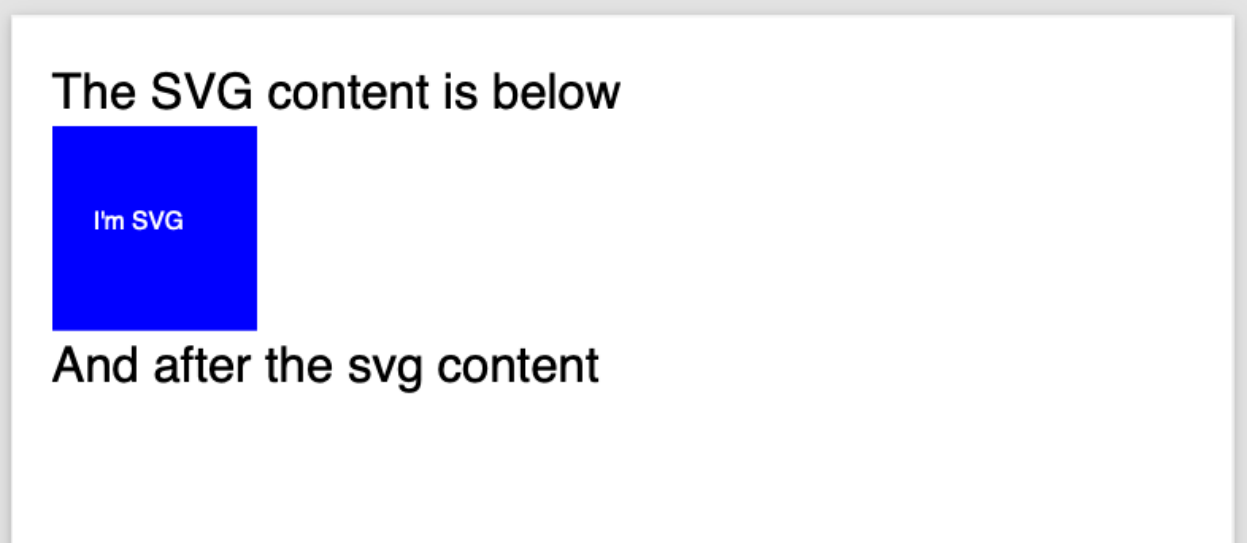
- ViewPorts

The drawing components should all be within a namespace qualified svg element, or prefixed svg at the document root.

There are may resources for SVG available, below is a description of the capabilities of scryber's implementation of SVG. And it is perfectly possible to draw an entire page in SVG within a body or section.

10.20.1 Drawing SVG content

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML >

<html xmlns='http://www.w3.org/1999/xhtml' >
<head>
    <style type="text/css">

        svg{
            font-family:sans-serif;
            font-size:12pt;
        }

        .colored {
            fill: blue;
        }

    </style>
</head>
<body style="padding:20pt;">
    <p>The svg content is below</p>
    <!-- Adding an svg rect and some text to the page -->
    <svg xmlns="http://www.w3.org/2000/svg" >
        <rect class="colored" width="100pt" height="100pt" >
        </rect>
        <text x="20" y="50" fill="white" >I'm SVG</text>
    </svg>
    <p>And after the svg content</p>
</body>
</html>
```



As can be seen, the svg content is as a block and renders within the flow of the content. The rect(angle) picks up the styles from the css and the font flows down from the svg container. The use of svg a an inline, or inline-block may be supported in the future.

With scryber it is possible to use svg elements directly in the document by delcaring a prefixed namespace at the top. And the example below will render the same. (See *Namespaces and their Assemblies* for more information on how namespaces are used)

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML >

<!-- Declare the naespace here -->
<html xmlns='http://www.w3.org/1999/xhtml'
    xmlns:svg="http://www.w3.org/2000/svg" >
<head>
    <style type="text/css">

        .canvas {
            font-family: sans-serif;
            font-size: 12pt;
        }
        .colored {
            fill: blue;
        }

    </style>
</head>
<body style="padding:20pt;">
    <p>The svg content is below</p>
    <div class="canvas">
        <!-- And prefix our elements here (in a div) -->
        <svg:rect class="colored" width="100pt" height="100pt" >
        </svg:rect>
        <svg:text x="20" y="50" fill="#EEF" >I'm SVG</svg:text>
    </div>
    <p>And after the svg content</p>
</body>
</html>
```

Note: depending on the purpose, this might be advantageous. But not make any html parsers happy unless wrapped in an svg:svg element.

All examples below will follow the standard <svg xmlns=" > convention.

10.20.2 Supported shapes

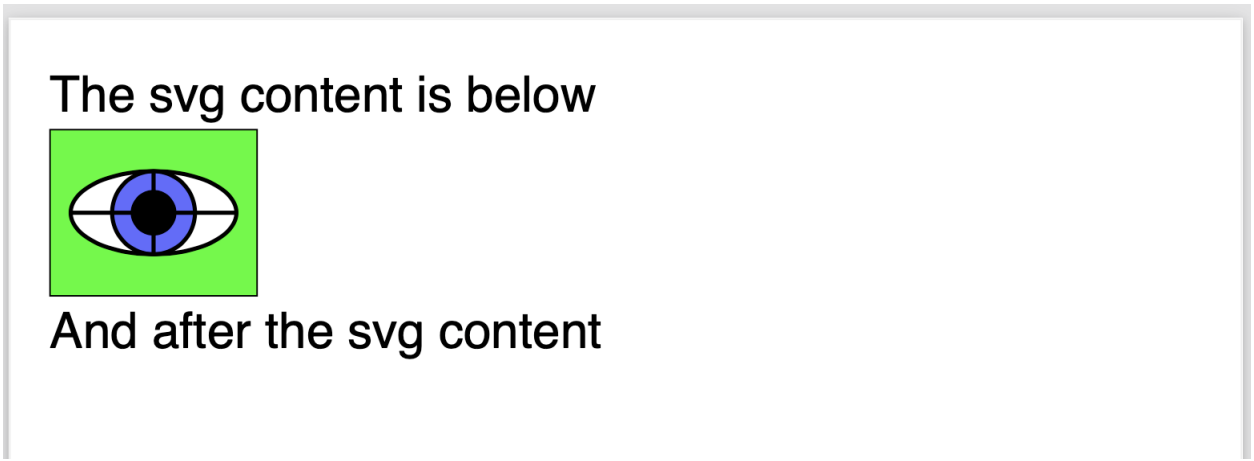
Scryber supports the standard shapes for rectangles, elipses, circles and lines. Generally, as closed shapes they will have a black fill and no stroke.

A group group (g) can contain multiple shapes and paths, and alter the style of inner content, e.g. applying a constitent stroke.

Without a width or height the svg element in scryber with size to the inner content, but it is good practice to specify values.

Scryber also supports the use of styles on the svg element itself.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML >

<html xmlns='http://www.w3.org/1999/xhtml' >
<head>
</head>
<body style="padding:20pt;">
    <p>The svg content is below</p>

    <svg xmlns="http://www.w3.org/2000/svg" style="border:solid 1px black" >
        <rect x="0pt" y="0pt" width="100pt" height="80pt" fill="lime" ></rect>
        <g id="eye" stroke="black" stroke-width="2pt" >
            <ellipse cx="50pt" cy="40pt" rx="40pt" ry="20pt" fill="white"></ellipse>
            <circle cx="50pt" cy="40pt" r="20pt" fill="#66F"></circle>
            <circle cx="50pt" cy="40pt" r="10pt" fill="black"></circle>
            <line x1="10" x2="90" y1="40" y2="40" />
            <line x1="50" x2="50" y1="20" y2="60" />
        </g>
    </svg>

    <p>And after the svg content</p>
</body>
</html>
```



10.20.3 Polylines, gons and paths

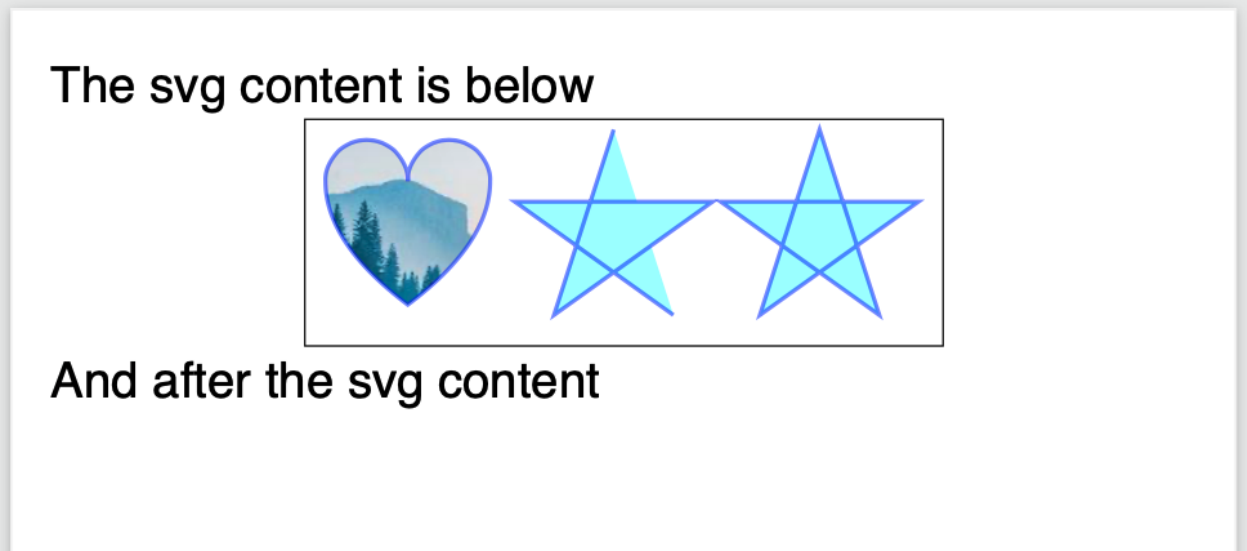
Scryber supports the standard paths, polylines and polygons

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML >

<html xmlns='http://www.w3.org/1999/xhtml' >
<head>
    <style type="text/css">

        .transparentish {
            fill: aqua;
            stroke:blue;
            stroke-width:2pt;
            fill-opacity: 0.5;
            stroke-opacity: 0.5;
        }

        .img-bg{
            fill: url(./images/landscape.jpg);
        }

    </style>
</head>
<body style="padding:20pt;">
    <p>The svg content is below</p>
    <div style="text-align:center;">

        <svg xmlns="http://www.w3.org/2000/svg" style="border:solid 1px black" width=
↪"310" height="110">
            <path class="transparentish img-bg" d="M 10,30
        A 20,20 0,0,1 50,30
        A 20,20 0,0,1 90,30
        Q 90,60 50,90
        Q 10,60 10,30 z" ></path>
            <polyline class="transparentish" points="150,5 121,95 198,40 102,40 179,95
↪" stroke="blue"
                    stroke-width="2pt"></polyline>
            <polygon class="transparentish" points="250,5 221,95 298,40 202,40 279,95
↪" stroke="blue"
```

(continues on next page)

(continued from previous page)

```
                    stroke-width="2pt"></polygon>
        </svg>
    </div>
    <p>And after the svg content</p>
</body>
</html>
```



A path has the operations explicitly defined within the 'd' attribute, see below.

A polyline is rendered using secific x,y points from the top left of the container.

A polygon automatically closes the path.

Scryber does not currently support the use of patterns or gradients as fills e.g. fill='url(#mypattern)', but does support images as fills, and backgrounds e.g. fill='url(./path/toimage.png)'. We will look at this for future releases.

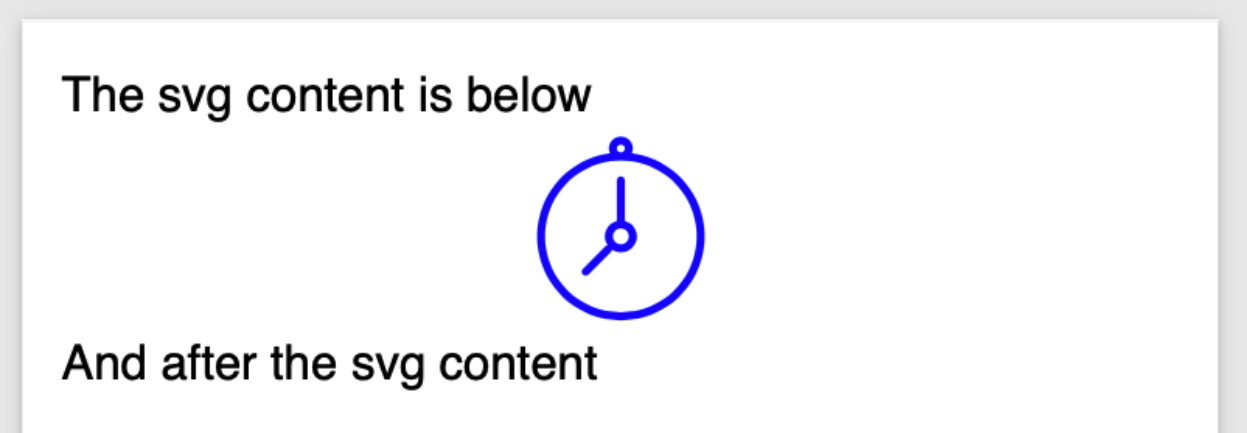
10.20.4 Drawing paths

Scryber supports the use of bezier paths for the creation of the complex curves and shapes. The format of the drawing data (d) is exacly the same as the **svg** drawing operations.

- M = moveto
- L = lineto
- H = horizontal lineto
- V = vertical lineto
- C = curveto
- S = smooth curveto
- Q = quadratic Bézier curve
- T = smooth quadratic Bézier curveto
- A = elliptical Arc
- Z = closepath

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML >

<html xmlns='http://www.w3.org/1999/xhtml' >
<head>
    <style type="text/css">

        .transparentish {
            fill: aqua;
            stroke:blue;
            stroke-width:2pt;
            fill-opacity: 0.5;
            stroke-opacity: 0.5;
        }

    </style>
</head>
<body style="padding:20pt;">
    <p>The svg content is below</p>
    <div style="text-align:center;">
        <svg id="ClockIcon" top="0" left="0" width="100" height="100" viewBox="0 0 20␣
↪20" xmlns="http://www.w3.org/2000/svg">
            <path fill="blue" d="M11.088,2.542c0.063-0.146,0.103-0.306,0.103-0.476c0-
↪0.657-0.534-1.19-1.19-1.19c-0.657,0-1.19,
            0.533-1.19,1.19c0,0.17,0.038,0.33,0.102,0.476c-4.085,0.535-7.243,
            4.021-7.243,8.252c0,4.601,3.73,8.332,8.332,8.332c4.601,0,
            8.331-3.73,8.331-8.332C18.331,6.562,15.173,3.076,11.088,2.542z M10,
            1.669c0.219,0,0.396,0.177,0.396,0.396S10.219,2.462,10,2.462c-0.22,
            0-0.397-0.177-0.397-0.396S9.78,1.669,10,1.669z M10,18.332c-4.163,
            0-7.538-3.375-7.538-7.539c0-4.163,3.375-7.538,7.538-7.538c4.162,0,
            7.538,3.375,7.538,7.538C17.538,14.957,14.162,18.332,10,18.332z M10.386,
            9.26c0.002-0.018,0.011-0.034,0.011-0.053V5.24c0-0.219-0.177-0.396-0.396-0.
↪396c-0.22,
            0-0.397,0.177-0.397,0.396v3.967c0,0.019,0.008,0.035,0.011,0.053c-0.689,0.
↪173-1.201,0.792-1.201,1.534c0,0.324,0.098,0.625,0.264,0.875c-0.079,0.014-0.155,0.
↪043-0.216,0.104l-2.244,2.244c-0.155,0.154-0.155,0.406,0,0.561s0.406,0.154,0.561,0l2.
↪244-2.242c0.061-0.062,0.091-0.139,0.104-0.217c0.251,0.166,0.551,0.264,0.875,0.264c0.
↪876,0,1.587-0.711,1.587-1.587C11.587,10.052,11.075,9.433,10.386,9.26z M10,11.586c-0.
↪438,0-0.793-0.354-0.793-0.792c0-0.438,0.355-0.792,0.793-0.792c0.438,0,0.793,0.355,0.
↪793,0.792C10.793,11.232,10.438,11.586,10,11.586z"></path>
        </svg>
        <!-- Icon from dribble -->
    </div>
    <p>And after the svg content</p>
</body>
</html>
```

The viewbox defines the area visible and will scale the content of the svg appropriately to the required width and height.



10.20.5 Line options

The stroke style also supports the standed ending and join options for paths, that will alter the way lines and vertices are rendered.

```
<path id="smiley" fill="yellow" stroke="black" stroke-width="8pt" stroke-linecap=
↪"round" stroke-linejoin="round"
              d="M50,10 A40,40,1,1,1,50,90 A40,40,1,1,1,50,10 M30,40 Q36,35,42,40 M58,
↪40 Q64,35,70,40 M30,60 Q50,75,70,60 Q50,75,30,60" />
```

10.20.6 Definitions and use

Scryber supports the definition of shapes and reuse within the content. This can either be directly, or within another viewbox for scaling and position.

The preserveAspectRatio is the standard svg enumeration that allows the content position in the viewbox to be defined on the outer container.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML>

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>

</head>
<body style="padding:20pt;">
    <p>The svg content is below</p>
    <div style="text-align:center;">
        <svg xmlns="http://www.w3.org/2000/svg">

            <!-- define our smiley approx 100 x 100 units -->
            <defs>
                <path id="smiley" fill="yellow" stroke="black" stroke-width="8pt"␣
↪stroke-linecap="round" stroke-linejoin="round"
                    d="M50,10 A40,40,1,1,1,50,90 A40,40,1,1,1,50,10 M30,40 Q36,35,42,
↪40 M58,40 Q64,35,70,40 M30,60 Q50,75,70,60 Q50,75,30,60" />
            </defs>

            <!-- use it in the middle so it is scaled to be fully visible -->
```

(continues on next page)

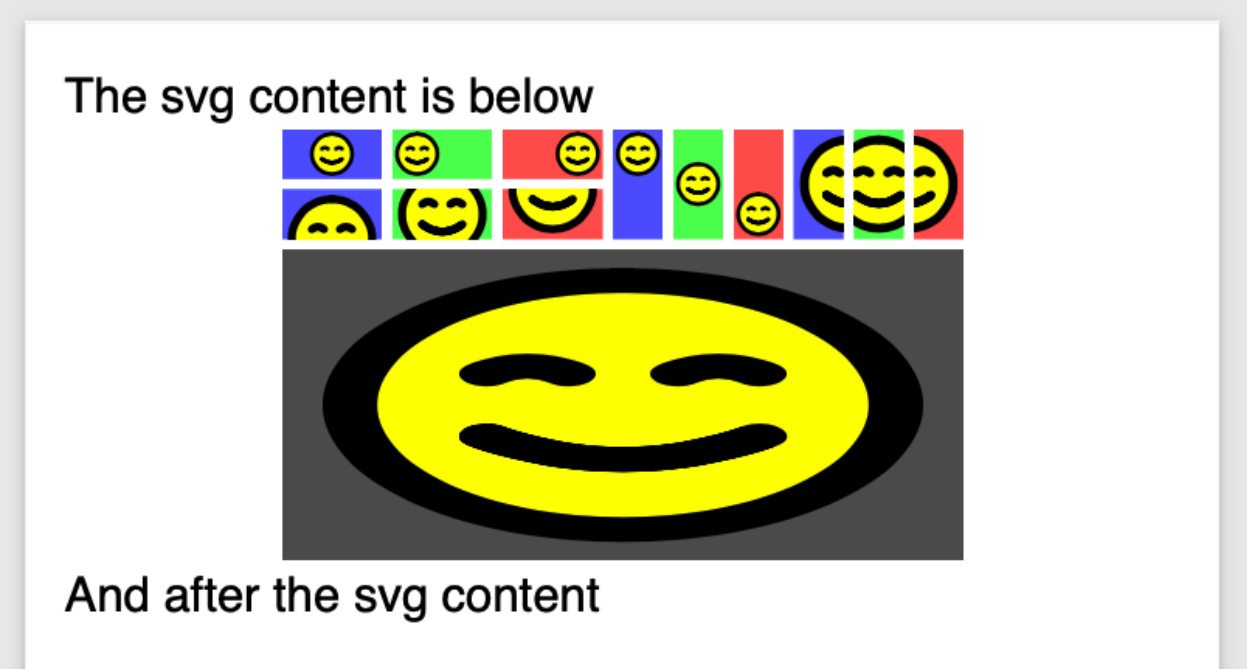
(continued from previous page)

```
            <svg id="smileyWrapper" x="0" width="50pt" height="25pt" viewBox="0 0 100␣
↪100" style="background-color:#5555FF;"
                preserveAspectRatio="xMidYMid">
                <use href="#smiley" />
            </svg>

            <!-- on the left side fully visible -->
            <svg id="smileyWrapper" x="55" width="50pt" height="25pt" viewBox="0 0␣
↪100 100" style="background-color:#55FF55;"
                preserveAspectRatio="xMinYMid">
                <use href="#smiley" />
            </svg>

            <!-- on the right side fully visible -->
            <svg id="smileyWrapper" x="110" width="50pt" height="25pt" viewBox="0 0␣
↪100 100" style="background-color:#FF5555;"
                preserveAspectRatio="xMaxYMid">
                <use href="#smiley" />
            </svg>

            <!-- slice will make the contents fill the box rather than fit -->

            <!-- top middle -->
            <svg id="smileyWrapper" x="0" y="30" width="50pt" height="25pt" viewBox=
↪"0 0 100 100" style="background-color:#5555FF;"
                preserveAspectRatio="xMidYMin slice">
                <use href="#smiley" />
            </svg>

            <!-- middle middle -->
            <svg id="smileyWrapper" x="55" y="30" width="50pt" height="25pt" viewBox=
↪"0 0 100 100" style="background-color:#55FF55;"
                preserveAspectRatio="xMidYMid slice">
                <use href="#smiley" />
            </svg>

            <!-- bottom middle -->
            <svg id="smileyWrapper" x="110" y="30" width="50pt" height="25pt" viewBox=
↪"0 0 100 100" style="background-color:#FF5555;"
                preserveAspectRatio="xMidYMax slice">
                <use href="#smiley" />
            </svg>

            <!-- meet vertical align -->

            <svg id="smileyWrapper" x="165" width="25pt" height="55pt" viewBox="0 0␣
↪100 100" style="background-color:#5555FF;"
                preserveAspectRatio="xMidYMin">
                <use href="#smiley" />
            </svg>

            <svg x="195" width="25pt" height="55pt" style="background-color:#55FF55;"␣
↪viewBox="0 0 100 100"
                preserveAspectRatio="xMidYMid meet">
                <use href="#smiley" />
            </svg>
```

(continues on next page)

(continued from previous page)

```
            <svg x="225" y="0" width="25pt" height="55pt" style="background-color:
↪#FF5555;" viewBox="0 0 100 100"
                preserveAspectRatio="xMidYMax meet">
                <use href="#smiley" />
            </svg>

            <!-- scale vertical align -->

            <svg id="smileyWrapper" x="255" width="25pt" height="55pt" viewBox="0 0␣
↪100 100" style="background-color:#5555FF;"
                preserveAspectRatio="xMinYMax slice">
                <use href="#smiley" />
            </svg>

            <svg x="285" width="25pt" height="55pt" style="background-color:#55FF55;"␣
↪viewBox="0 0 100 100"
                preserveAspectRatio="xMidYMax slice">
                <use href="#smiley" />
            </svg>

            <svg x="315" y="0" width="25pt" height="55pt" style="background-color:
↪#FF5555;" viewBox="0 0 100 100"
                preserveAspectRatio="xMaxYMax slice">
                <use href="#smiley" />
            </svg>

            <!-- Finally just fill the box -->

            <svg x="0" y="60" width="340pt" height="155pt" style="background-color:
↪#555555;" viewBox="0 0 100 100"
                preserveAspectRatio="none">
                <use href="#smiley" />
            </svg>

        </svg>
    </div>
    <p>And after the svg content</p>
</body>
</html>
```



10.20.7 SVG Text

Scryber supports the use of the SVG Text and text spans for rendering characters within the drawing.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML>

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <link type="text/css" rel="stylesheet" href="https://fonts.googleapis.com/css2?
↪family=Roboto:ital,wght@0,100;0,700;1,100&amp;display=swap" />
    <style>

        body {
            font: 12pt 'Roboto';
            padding: 20pt;
        }

        .small {
            font: italic 13px 'Roboto';
        }

        .heavy {
            font: bold 30px 'Roboto';
        }

        .red {
            font: italic 40px 'Roboto';
            fill: red;
        }

    </style>
</head>
```

(continues on next page)

(continued from previous page)

```
<body style="padding:20pt;">
    <p>The svg content is below</p>
    <div style="text-align:center;">
        <svg xmlns="http://www.w3.org/2000/svg">
            <text x="20" y="35" class="small">My</text>
            <text x="35" y="35" class="heavy">cat</text>
            <text x="55" y="60" class="small">is</text>
            <text x="60" y="60" class="red">Grumpy!</text>
        </svg>
    </div>
    <p>And after the svg content</p>
</body>
</html>
```

Here we are linking to and using the Roboto font from the google api's.

10.20.8 Referencing drawings

It is also possible to load an svg file directly into the document with an embed option.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML>

<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
    <link type="text/css" rel="stylesheet" href="https://fonts.googleapis.com/css2?
↪family=Roboto:ital,wght@0,100;0,700;1,100&amp;display=swap" />
    <style>

        body {
            font: 12pt 'Roboto';
            padding: 20pt;
        }

        .small {
            font: italic 13px 'Roboto';
        }
```

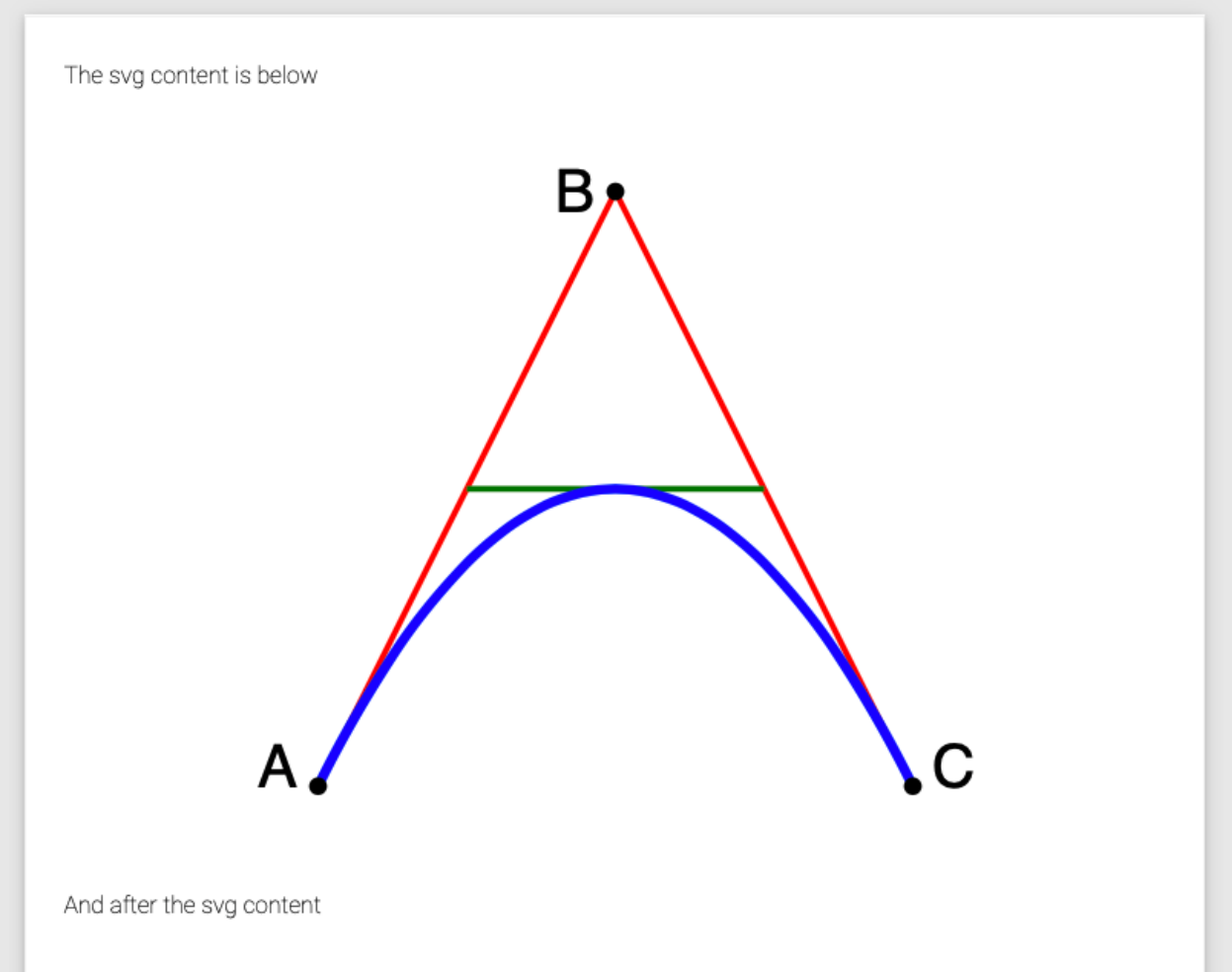
(continues on next page)

(continued from previous page)

```
        .heavy {
            font: bold 30px 'Roboto';
        }

        .red {
            font: italic 40px 'Roboto';
            fill: red;
        }

    </style>
</head>
<body style="padding:20pt;">
    <p>The svg content is below</p>
    <div style="text-align:center;">
        <embed src="./Fragments/MyDrawing.svg" />
    </div>
    <p>And after the svg content</p>
</body>
</html>
```

And the referenced SVG file is ./Fragments/MyDrawing.svg

```
<svg xmlns="http://www.w3.org/2000/svg" width="500" height="400">

    <path id="lineAB" d="M 100 350 l 150 -300" stroke="red" stroke-width="3" fill=
↪"none" />
    <path id="lineBC" d="M 250 50 l 150 300" stroke="red" stroke-width="3" fill="none
↪" />
    <path d="M 175 200 l 150 0" stroke="green" stroke-width="3" fill="none" />
    <path d="M 100 350 q 150 -300 300 0" stroke="blue" stroke-width="5" fill="none" />
    <!-- Mark relevant points -->
    <g stroke="black" stroke-width="3" fill="black">
        <circle id="pointA" cx="100" cy="350" r="3" />
        <circle id="pointB" cx="250" cy="50" r="3" />
        <circle id="pointC" cx="400" cy="350" r="3" />
    </g>
    <!-- Label the points -->
    <g font-size="30" font-family="sans-serif" fill="black" stroke="none" >
        <text x="70" y="350" >A</text>
        <text x="220" y="60" >B</text>
        <text x="410" y="350" >C</text>
    </g>
</svg>
```



10.20.9 Attributes Supported

Only a few of the full capabilities and attributes of SVG are supported. More are supported using the style=" css attribute settings, and we will be adding more in future.

10.21 Document outline (or bookmarks)

Structing a document also allows for the outline. This is effectively a table of contents available in PDF readers that support quick navigation of the whole document, to go to the section or content needed.

In Scryber the title attribute is converted to display as the outline of the document. Nested content will be under any parent title.

10.21.1 Simple Outline

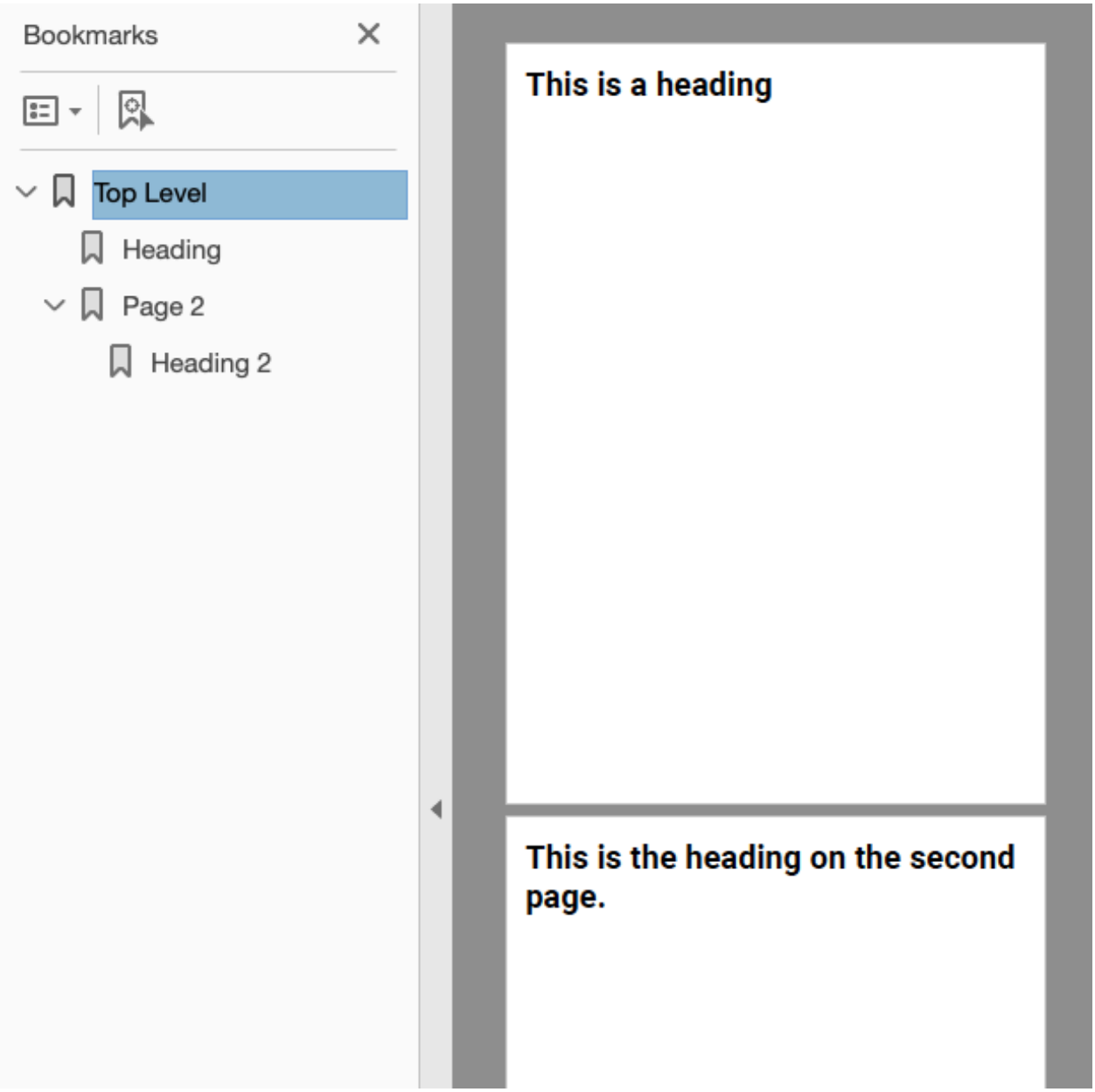
```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE HTML>
```

(continues on next page)

(continued from previous page)

```
<html xmlns='http://www.w3.org/1999/xhtml' >
<head>
    <link type="text/css" rel="stylesheet" href="https://fonts.googleapis.com/css2?
↪family=Roboto:ital,wght@0,100;0,700;1,100&amp;display=swap" />
    <style>

        body {
            font: 12pt 'Roboto';
            padding: 20pt;
        }

    </style>
</head>
<body title="Top Level" style="padding:20pt;">
    <div>
        <h1 title="Heading" >This is a heading</h1>
    </div>

    <section title="Page 2" >
        <h1 title="Heading 2" >This is the heading on the second page.</h1>
    </section>
</body>
</html>
```

When output the reader application or browser can show the content of the outline. Selecting any of the bookmark items should navigate directly to the page the content is on.



10.22 Securing Documents

Scryber supports securing documents with restrictions (printing, copying and modifying) along with password protection.

10.22.1 Restrictions

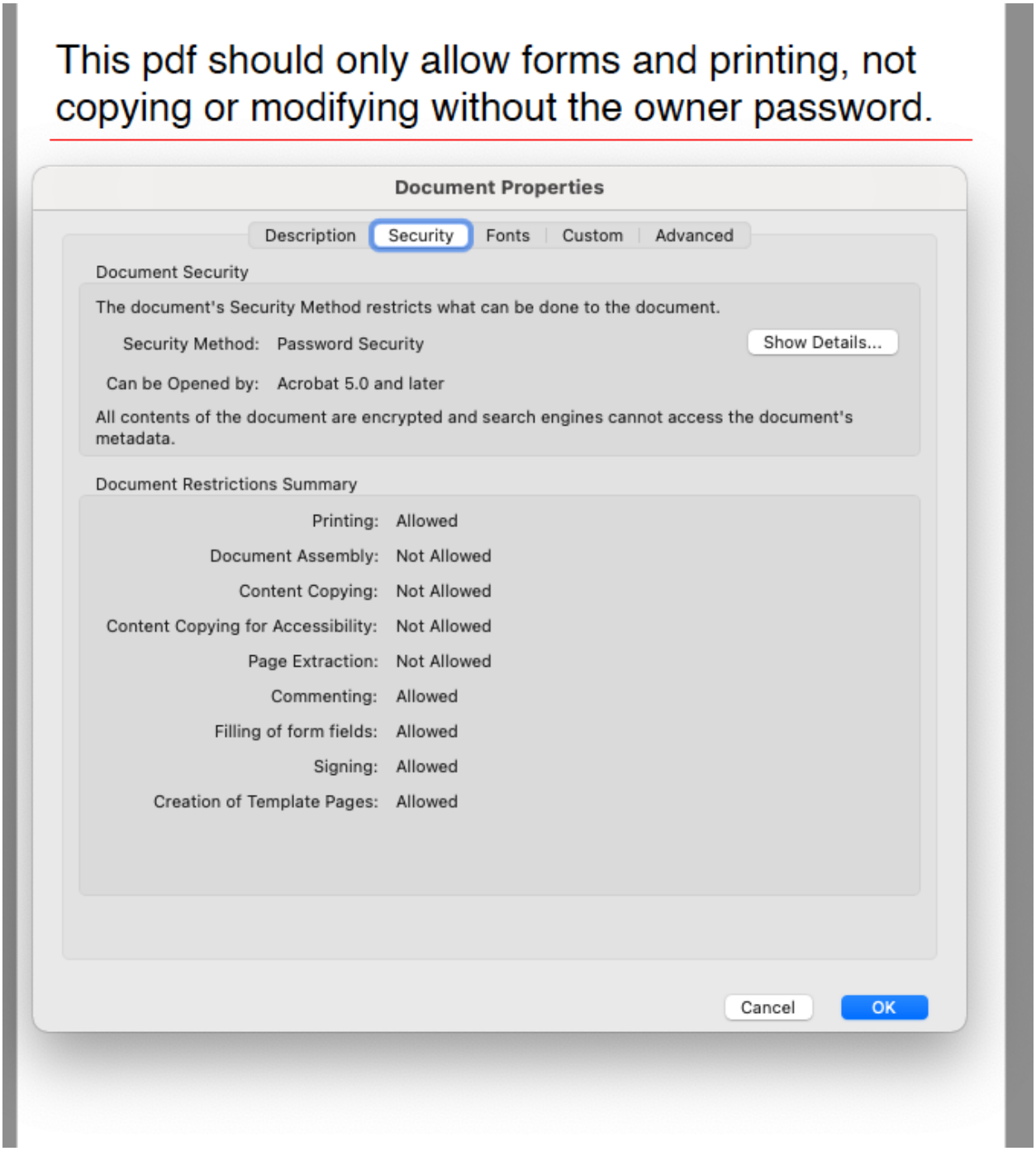
The document restrictions are managed through the header meta tag with the name print-restrictions or print-encryption.

```
<meta name="print-restrictions" content="allow-forms allow-printing"  />
<!-- optional print encryption settings -->
<meta name="print-encryption" content="128bit" />
```

These restrictions can be applied directly to a document without an owner password if the parser-mode is Lax (default). However this will be a randomly generated password, and unretrieveable. (Although the document will still open and be secured).

If the parser-mode is strict then an error will be thrown. See (*Scryber Trace Log*)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd">

<html lang='en' xmlns='http://www.w3.org/1999/xhtml'
    title="Root">
<head>
    <meta charset='utf-8' name='author' content='Richard Hewitson' />
    <title>{@:title}</title>
    <meta name="print-restrictions" content="allow-forms allow-printing" />
    <meta name="print-encryption" content="128bit" />
</head>
<body class="grey" title="Outer">
    <p class='highlight' title="Inner"
    style="margin:20pt; border-bottom:solid 1px red; padding:4px; margin-top:20pt;␣
↪font-family:sans-serif;  ">
    This pdf should only allow forms and printing, not copying or modifying without␣
↪the owner password.</p>
    <p>{@:title}</p>
</body>
</html>
```



10.22.2 Restriction values

The following values are supported on the print-restrictions meta tag.

- allow-printing - The end user will be able to print the document.
- allow-accessibility - The end user can use accessibility tools to read the document.
- allow-annotations - The end user can add annotations and comments.

- allow-copying - The end user can copy content from the document
- allow-modifications - The end user can modify the document and add remove pages etc.
- allow-forms - The end user can complete forms, and sign content.
- all - All restrictions are applied.

The print-encryption values supported are

- 40bit - lowest level, and most basic restrictions.
- 128bit - more secure and supports enhanced restrictions (default if no print-encryption is set).

10.22.3 Adding Passwords

The best use of the document restrictions is when providing password(s) at generation time.

There are 2 passwords used

- owner - to unlock the restrictions and allow all actions.
- user - to allow opening of the document.

They can both be the same, if wanted, and are set on the Document.PasswordProvider property, through the implementation of an IPDFSecurePasswordProvider

The quickest set up is to use the DocumentPasswordProvider in Scryber.Secure. This can be initialized with one or two strings, or secure strings.

```
using (var doc = Document.ParseDocument(path))
{
    doc.PasswordProvider = new Scryber.Secure.DocumentPasswordProvider("Password",
↪"Password");
    doc.Params["title"] = "Hello World";

    using (var stream = DocStreams.GetOutputStream("ProtectedHtml.pdf"))
    {
        doc.SaveAsPDF(stream);
    }

}
```

Warning: The use of simple strings as in memory passwords onto documents could be a security risk, and the IDisposable System.Secure.SecureString is more appropriate.

10.22.4 Implementing the IPDFSecurePasswordProvider

The secure password provider interface is a trivial matter of implementing a class that can set some document password settings, based on the document path. The way they are loaded and how they are retuned is up to the implementor.

If the provider returns false then the document will not be secured, otherwise it will use the IDocumentPasswordSettings as needed.

```
//using System.Security

public interface IPDFSecurePasswordProvider : IDisposable
```

(continues on next page)

(continued from previous page)

```
{
    /// <summary>
    /// Implementers should use this method to load specific security settings pased␣
↪on a specific path and return required values
    /// </summary>
    /// <param name="documentpath">The source path the document was loaded from</
↪param>
    /// <param name="settings">Set to the security settings to be associated with␣
↪this document if IsSecure returns true.</param>
    /// <returns>Return true to assign security settings, or false to not.</returns>
    bool IsSecure(string documentpath, out IDocumentPasswordSettings settings);

}


public interface IDocumentPasswordSettings : IDisposable
{
    /// <summary>
    /// Gets the Owner Password for the document.
    /// If not set, then it must be set in code before a secure document is output
    /// </summary>
    public SecureString OwnerPassword { get; }

    /// <summary>
    /// Gets the User Password for the document. If null then NO password is required␣
↪to open and view the document
    /// </summary>
    public SecureString UserPassword { get; }

}
```

10.23 To code or not to code...

Scryber does not rely on xml / html, but it makes life easier and is more visual and structured.

When ever you parse a Document or component you are simply creating the same as you could in code.

10.23.1 XML Template

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd">

<html xmlns='http://www.w3.org/1999/xhtml' >

<head>
    <title>HTML Document</title>
    <style>
        .grey{ background-color: grey; }
    </style>
</head>

<body class="grey" title="Page 1">
    <p title="Inner">Hello World, from scryber.</p>
```

(continues on next page)

(continued from previous page)

```
</body>
</html>
```

10.23.2 The same in code

```
var doc = new Document();
doc.Info.Title = "Coded Document";

var style = new StyleDefn(".grey");
style.Background.Color = (PDFColor)"grey";
doc.Styles.Add(style);

var pg = new Page();
pg.StyleClass = "grey";
pg.OutlineTitle = "Page 1";

var para = new Paragraph();
para.Contents.Add(new TextLiteral("Hello World From scryber"));

pg.Contents.Add(para);
doc.Pages.Add(pg);

return doc.ProcessDocument();
```

10.23.3 The same in XLinQ

```
XNamespace html = "http://www.w3.org/1999/xhtml";
XElement root = new XElement(html + "html",
    new XElement(html + "head",
        new XElement(html + "title", "XElememt Document"),
        new XElement(html + "style", ". grey { background-color: gray;}")
    ),
    new XElement(html + "body",
        new XAttribute("class", "grey"),
        new XAttribute("title", "Page 1"),
            new XElement(html + "p",
            new XAttribute("title", "Inner"),
            new XText("Hello World from Scryber"))
        )
    );

string basePath = string.Empty;
doc = Document.ParseDocument(root.CreateReader(), basePath, ParseSourceType.
↪DynamicContent);
```

Note: The use of the base path allows relative images and styles to be used from a reader in dynamic content.

10.23.4 Loading partial content

Scryber supports the loading of partial content (not a whole document) through the use of the instance method ParseTemplate(), or the Document static Parse method.

There are a number of overloads for the parse template method that use streams, text readers and xml readers. But as a simple loading mechanism it can be useful to include some custom stored content.

```
//This content can be loaded from any source.

var content = "<p xmlns='http://www.w3.org/1999/xhtml' >" +
                "This <b>Is my content</b>" +
                "</p>";

using (var reader = new StringReader(content))
{
    var comp = doc.ParseTemplate(doc, reader) as Component;
    (doc.Pages[0] as Page).Contents.Add(comp);
}
```

The use of the first component argument in ParseTemplate is to provide the source path for any relative references. It can also be called with the owner and a base path.

10.23.5 Resolving paths to custom content

The static Parse method has 12 overloads and counting, from a simple stream and base path, to the explicit PDFGeneratorSettings. It also supports the use of the PDFReferenceResolver, that can implement custom methods to resolve references to content (e.g. database images, or authenticated document sources).

```
//custom reference resolver implementation

private IPDFComponent ResolveReference(string filename, string xpath,␣
↪PDFGeneratorSettings settings)
{
    Stream content = GetMyContentForPath(filename);
    return Document.Parse(filename, content, ParseSourceType.DynamicContent, settings.
↪Resolver);
}
```

And this method can be used when parsing inner content or documents with references to other content.

```
using (var reader = new StringReader(content))
{
    Document.Parse("", reader, ParseSourceType.DynamicContent, new␣
↪PDFReferenceResolver(this.ResolveReference));
}
```

10.23.6 Why use one over the other

We always think that the declarative is better for what you need, but sometimes building in code works. See the *Class Hierarchy* for a break down of the class heierarchy.

In this documentation, we will concentrate on the use of the declarative html with code where appropriate, but remember that everything that is declared can be coded too.

10.24 Class Hierarchy

When working with the code or adding objects to the code once parsed it is good to understand the hierarchy.

Scryber has a top level of Components. These create the basic level of document structure.

10.24.1 Base Component Classes

The base classes form the foundation of the functionality for each of the main concrete classes. It's much easier to create your own funtionality using one of these classes.

- Component - Implements the IPDFComponent interface along with IPDFBindable and
- ContainerComponent - Holds child instances. Implements the 'InnerContent' collection as a protected property and the IPDFContainer.Contents implementation. Also passes the lifecycle methods to children.
- **VisualComponent - Extends a container to be styled with the IPDFStyledComponent interface and a lot of properties for i**
  - Panel - Base class for the standard container componnents (Div, Span, etc) that implements the IPDFViewPortComponent interface to do the laying out of content.
  - PageBase - Base class for all page types.
  - ImageBase - Base class for all the image types.
  - TextBase - Base class for the textual components and implements the IPDFTextComponent interface.
  - ShapeComponent - Base class for drawing components, implementing the IPDFGraphicPathComponent interface.

All components inheriting from VisualComponent have a virtual method for GetBaseStyle() which returns the default style for that component before anything is applied.

The PDFObjectType used in constructors is simply a 4 character struct that can identify the type of component, and can also be used for generating ID's. It can be directly cast from a string value or const string.

Note: Any custom classes should include a parameterless constructor if they sould be parsed as part of some other xml/xhtml content.

10.24.2 Collection classes

Scryber maintains a bi-directional structure to the document content graph. When a component is added as a child to a container, that child's Parent value is updated to the container. As such, each child knows it's parent, page and document.

In order to maintian this the ContainerComponent creates the ComponentList class with itself as an owner.

Scryber then provides the abstract ComponnetWrappingList and ComponentWrappingList<T> classes for stronger typing on the contents of the collection and implementing the ICollection<T> interface.

e.g. The TableGrid has a Rows property of TableRowCollection which inherits from ComponentWrappingList<TableRow> and wraps the ContainerComponent.InnerContent.

```
public class TableGrid : ContainerComponent
{
    //Strongly typed collection of Rows that will have their parent set automatically.
    public TableRowList Rows
    {
        get
        {
            if (this._rows == null)
                this._rows = new TableRowList(this.InnerContent);
            return this._rows;
        }
    }

}

//The strongly typed collection for TableRows.
public class TableRowList : ComponentWrappingList<TableRow>
{
    public TableRowList(ComponentList content)
        : base(content)
    {
    }
}
```

10.24.3 Concrete Component Classes

All the components are in the Scryber.Components namespace and inherit from VisualComponent

- **PageBase**
  - **Page - a single page by default style, with a Header and Footer.**
    - Section - allows multiple layout pages by default, with a continuation header and footer.
  - PageGroup - a set of PageBases instances but also a Header, Footer and continuation header and footer that are passed down.
- **Panel**
  - Div - basic concrete panel implementation with contents and a full width.
  - BlockQuote - basic block quote implementation with a custom style of 10pt margins.
  - Canvas - a container where all content will be relatively positioned and the content clipped to the canvas bounds.
  - **List - contains inner list items.**
    - ListOrdered - a list that has a default decimal numbering.
    - ListUnordered - a list that has a default bullet addornment.
    - ListDefinition - a list that has terms and content.
  - ListItem - the individual items in a list.
  - UserComponent - allows the dynamic loading of content from a remote source.
  - Paragraph - a block of inner content, with a 4pt margin at the top as a default style.
  - Preformatted - a block of inner content, with a default style for rendering code.

- **TextBase**
  - Date - renders the current of defined date in a specific format.
  - Number - renders a numeric value in any specific format.
  - PageNumberLabel - renders the current page (along with totals) in any specific format.
  - PageOfLabel - renders the page number of another component.
- TextLiteral - A non-visual component for text strings, including assigment within the constructor.
- TableGrid - A layout of content in a tabular way.
- TableRow - A single row of cells within a grid.
- TableCell - the final content of the cells in a table grid.
- **ShapeComponent**
  - HorizontalRule - basic flat line.
  - Line - Line that supports a position and size.
  - Path - Complex path definition with M(oves), L(ines to) etc.
  - **PolygonBase**
    - Polygon - Multi-sided shape with style.
    - Rectangle - A 4 sided shape with style.
    - Triangle - Just the 3 sides.
    - Ellipse - A box bounded circle or ellipse with style.
- PageBreak - Forces the flow onto the next page if possible.
- ColumnBreak - Forces the flow onto the next column or page if possible.
- LineBreak - Forces the flow onto a new line.

10.24.4 Html Classes

When parsing content from HTML the document component graph will be constructed from subclasses of the main components in the Scryber.Html.Components namespace.

```
namespace Scryber.Html.Components {

    [PDFParsableComponent("div")]
    public class HTMLDiv : Scryber.Components.Div
    {

    }
}
```

10.24.5 Layout content

In the creation of a PDF document, the components above are used to create the actual layout items. These are much more basic, but know how to generate the pdf content streams and data used by PDF readers.

If a document has a Page, and then a Section with 2 page breaks - the layout will be 4 pages long with all the text and runs in the respective pages.

If needed any component can implement or override the IPDFViewPortComponent interface and return a new LayoutEngine for that component. The LayoutEngineBase and LayoutEnginePanel are good starting points to layout your own custom content.

- PDFLayoutDocument - Top level holding font references, image resource references and the list of layout pages.
- PDFLayoutPage - A single page of a content block, with an optional header content block and or footer content block, and any absolutely positioned regions.
- PDFLayoutBlock - A grouping of one or more column regions along with any relatively positioned regions, that will render the style.
- PDFLayoutRegion - A single continuous set of lines and/or other blocks.
- PDFLayoutLine - A single line of content runs.
- **PDFLayoutRun - A single lightweight atomic graphical content operation.**
  - **PDFTextRun - Textual operation**
    - PDFTextRunBegin - Start of the text, includes setting the font etc.
    - PDFTextRunCharacter - Text Drawing operation
    - PDFTextRunNewLine - Simple line break operation
    - PDFTextRunProxy - Placeholder for text to come from the owning component.
    - PDFTextRunEnd - Completion of text.
    - PDFTextRunSpacer - Offset of a line run to allow for other content.
  - PDFLayoutXObject - Renders PDF content as a separate stream, return the reference to that stream.
- PDFLayoutComponentRun - allows the owning component to render it's own content explicitly (e.g. Paths).

10.24.6 Content Styles

The style classes are based around a dictionary of inherited and direct style item keys with storongly typed style value keys. All of the standay ones are defined in the Scryber.Styles.StyleKeys static class.

If a style value is inherited, the it will be copied to any descendent element (e.g. FontFamily) and any direct value will only be used on the component it is defined on (e.g. BackgroundColor)

Implementor can create their own style items and keys as needed using the static constructor methods with distinct object types (use mixed case to ensure they are unique).

```
const bool INHERITED = true;
var tocStyle = StyleKey.CreateStyleItemKey((PDFObjectType)"Ctoc", INHERITED);
var tocLeader = StyleKey.CreateStyleValue<LineStyle>((PDFObjectType)"Ctld", tocStyle);
```

This can then be used on any style definition or styled component to get or set a value, it can be bound to a value, and as it is inherited, will flow down with the content (merged).

```
var styleDefn = new StyleDefn();
styleDefn.SetValue(tocLeader, LineStyle.Dotted);

LineStyle default = LineStyle.None;
var defined = styleDefn.GetValue(tocLeader, default);

if(styleDefn.TryGetValue(tocLeader, out defined)
{
```

(continues on next page)

(continued from previous page)

```
    //Do something with the defined style.
}
```

The style class hierarchy is as follows.

- **StyleBase - root abstract class that holds the actual values.**
  - **Style** [StyleBase - the main class used on components themselves directly.]
    - StyleDefn : Style - has a class matcher property that will ensure that this style is only applied to Components that match.
    - StyleFull : Style - a readonly, locked set of style values with known values - position, font, padding etc.
  - StyleGroup : StyleBase - a collection of style base items, that can be treated as one item in an outer collection.

The document has a Styles property which is a StyleCollection, so any of the above can be added to the the document. Each VisualComponent has a Style property where these values can be directly applied.

The flow for creating a full style for a component is linear.

1. The GetBaseStyle returns a new instance the standard style for a component. 1. If the component inherits from a super class VisualComponent then it should call the base.GetBaseStyle() and apply any styles to that before returning. 1. The GetAppliedStyle is then called with the base style. 1. This traverses up the component hierarchy, finally reaching the document. 1. The document calls MergeInto on its style collection with the base style. 1. Each style within the collection is MergedInto the style. 1. If that style is a StyleDefn it is checked to make sure it is matched, before being merged. 1. If that style is a StyleGroup, the it calls MergeInto on its own collection of styles. 1. If it should be merged, then each style value is assessed to see if it exists and compares the priority. 1. If the style that should be merged is a higher priority then the value is replaced. 1. We then come back to the original component and any direct styles are applied to to orginal base. 1. Once this is done it is pushed onto the StyleStack, where the hierarchy of styles from parent components are. 1. And finally a full style is built based on inherited and direct values. 1. That full style is retained and used through the rest of the layout and rendering.

Despite the number of steps, the build of styles is usually not an issue, compared to extracting font files, image binary data or encrypting streams. However for some documents with a large number or containers e.g. a very long table with many rows it can become the limiting factor as well as memory intensive.

The template element automatically caches the style for each of the inner contents, rather than building every time. This can speed the generation, but if it causeing issues can be switched off using the data-cache-styles=false attribute. This will force the styles to be built each and every time.

```
<templalate data-bind='{@:Model.Items}' data-cache-styles='false' >
    <tr>
        <td class='desc-cell' >{@:.Description}</td>
        <!-- can be applied individually so that they are cached -->
        <td class='val-cell' data-style-identifier='boundcellValue' >{@:.Value}</td>
    </tr>
</template>
```

10.24.7 Why and when to implement

A lot of the time, it is easier to use compound components to build all the main characteristics of the content needed. However sometimes there is a need to use explict functionality or capabilities that are not currently available.

At scryber we also use this framework extensively to provide new top level features with safe knowledge the lower engine layers can deal with the grunt work.

See *Scryber Trace Log* and *Extending Scryber - td* along with *Namespaces and their Assemblies* for more on this.

10.25 Scryber Trace Log

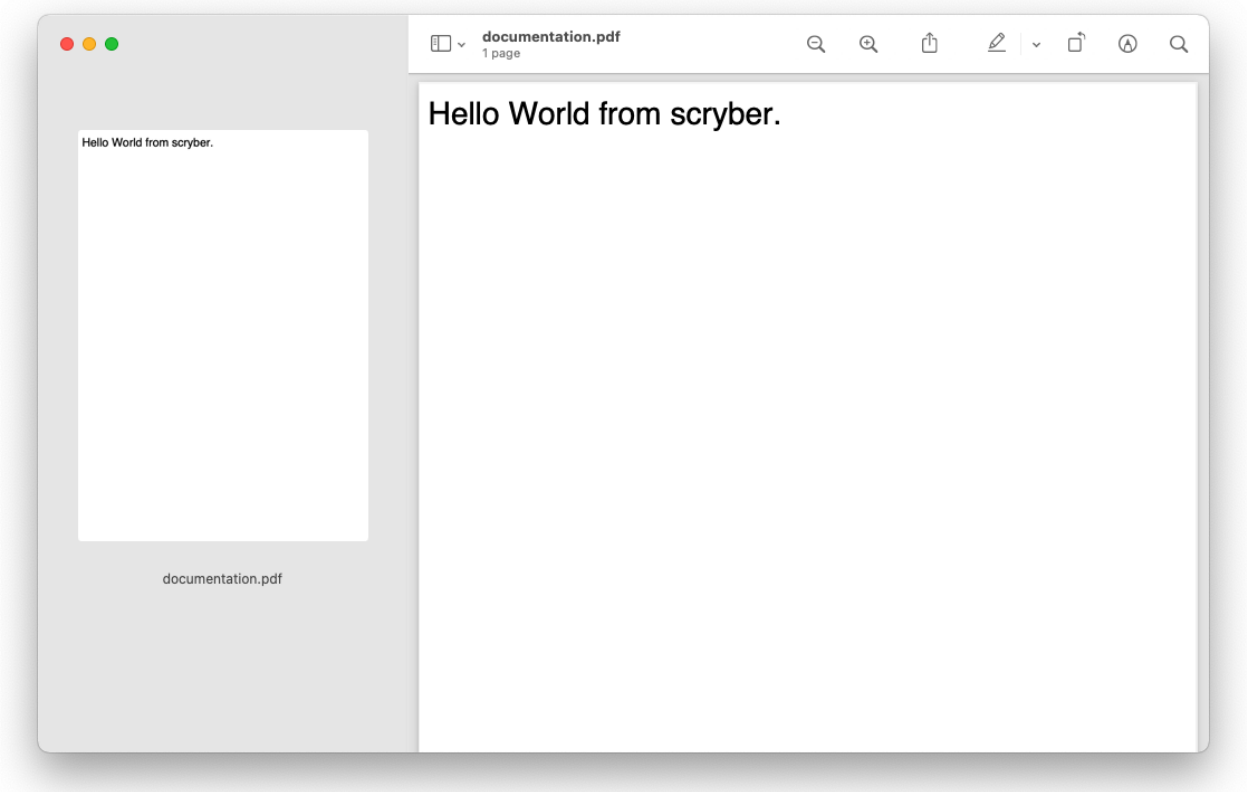
Internally scryber has lots of logging that can be used to understand what is going on underneath, without having to debug the libraries This is achieved with the scryber processing instruction that needs to be at the top of the file.

10.25.1 Example without logging

If we have the below example for our template.

```
<!DOCTYPE HTML >
<html xmlns='http://www.w3.org/1999/xhtml' >
    <head>
        <title>Hello World</title>
        <link href='../css/printstyles.css' rel='stylesheet' />
    </head>
    <body>
        <div style='padding:10px; text-align:center'>
            <img src='../images/sitelogo.png'>
            <p>Hello World from scryber.</p>
        </div>
    </body>
</html>
```

Then when we generate, we will get the following output



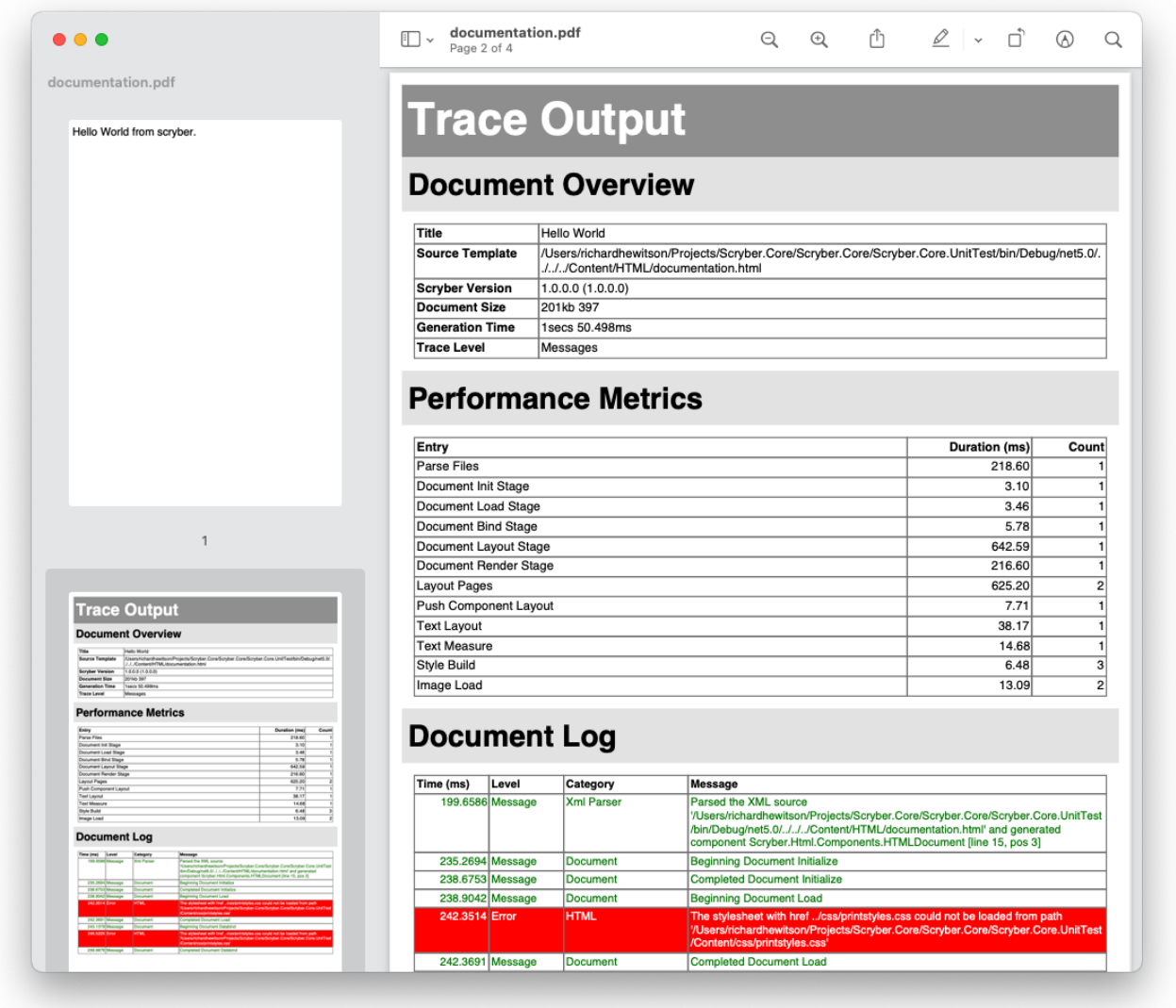
It is there, but there is no image, and it's not looking like we expected.

However if we add our scryber processing intruction to append the log for messages then we see so much more going on.

10.25.2 Example with logging

```
<?scryber append-log='true' log-level='Messages' parser-log='true' ?>
<!DOCTYPE HTML >
<html xmlns='http://www.w3.org/1999/xhtml' >
    <head>
        <title>Hello World</title>
        <link href='../css/printstyles.css' rel='stylesheet' />
    </head>
    <body>
        <div style='padding:10px; text-align:center'>
            <img src='../images/sitelogo.png'>
            <p>Hello World from scryber.</p>
        </div>
    </body>
</html>
```

Then when we generate, we will get the following output



Here we can see that there are another 3 pages added to the document - and quite a few errors. Specifically the stylesheet could not be found, and nor could the image.

10.25.3 Scryber processing instruction

The following are the supported options on the processing instruction.

- **'append-log' - Controls the tracing log output for a single document**
  - false - This is the default and the document will be rendered as normal.
  - true - If set to true, then once the document has been generated, a trace log of output will be appended to the resultant file, containing all the recorded entries.
- **'log-level' - This is an enumeration of the granularity of the logging performed on the pdf file. Values supported (from lea**
  - Off - no entries be recorded.
  - Errors - only errors will be recorded (depending on the parser mode switch)

  - Warnings - warnings will occur if some of the contents cannot be loaded, or the parsing fails for a non-error condition.
  - Messages - This will output key stage messages for the generation of the file.
  - Verbose - A quantity of messages will be output for each of the compoents, and is a useful level to understand what is going wrong (if anything) with your document.
  - Diagnostic - This will generate a large log file and can slow the creation of a PDF file significnatly. But it's very informative.

- **'parser-log' - Controls the logging from the xml parser.**
  - true - then both the reading of the content, to create the document, as well as the output of the content to PDF will be recorded.
  - false - then only messages from the content creation and output will be recorded.

- **'parser-culture' - specifies the global culture settings when parsing a file for interpreting dates and number formats in the**
  - en-gb - This specifies the english, britsh culture. It can be useful for reading number formats or dates from files e.g.
  - es-es - This will read spanish nuber formats where . 'dot' is a thousand separator and , 'comma' is the decimal separator.

- **'parser-mode' - Defines how errors will be recorded if unknown or invalid attributes values are encountered.**
  - Strict - Will raise exceptions to the top of the stack and must be handled in your code. (Good for dev)
  - Lax - Default. If this is set then the parser is more complianant, where errors will be logged, but not cause the output to fail. (Good for Prod).

Note: If you set the log level to Diagnostic for the Hello World example, the appended log file is around 10 pages in length. If it's a long document - diagnostic is going to hurt.

10.25.4 Tracing Details

There is some really good information available in the tracing output not just in the logging, but also on the metrics and overview.

Trace Output

Document Overview

Title	Hello World
Source Template	/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/documentation.html
Scryber Version	1.0.0.0 (5.0.0.4)
Document Size	201kb 397
Generation Time	2.489ms
Trace Level	Messages

Performance Metrics

Entry	Duration (ms)	Count
Parse Files	0.41	1
Document Init Stage	0.01	1
Document Load Stage	0.00	1
Document Bind Stage	0.04	1
Document Layout Stage	0.25	1
Document Render Stage	0.71	1
Layout Pages	0.23	2
Push Component Layout	0.01	1
Text Layout	0.05	1
Text Measure	0.02	1
Style Build	0.03	2

Document Log

Time (ms)	Level	Category	Message
000.1451	Message	Xml Parser	Parsed the XML source '/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/documentation.html' and generated component Scryber.Html.Components.HTMLDocument [line 13, pos 3]
001.4105	Message	Document	Beginning Document Initialize
001.4216	Message	Document	Completed Document Initialize
001.4223	Message	Document	Beginning Document Load
001.4241	Message	Document	Completed Document Load
001.4280	Message	Document	Beginning Document Databind
001.4734	Message	Document	Completed Document Databind
001.5227	Message	Document	Beginning Document layout
001.5455	Message	Document Layout	Starting the layout of page sect1
001.5653	Message	Layout Engine	Laying out the page component 'sect1' at page index 0
001.7595	Message	Document Layout	Completed the layout of page sect1 and now on page index 0
001.7740	Message	Document	Completed Document layout
001.7877	Message	Document	Beginning output of the document with render options : PDF Compliance = None, Compression = FlateDecode, Names = ExplicitOnly
001.7895	Message	Layout Document	Outputting document to the PDFWriter

The top section will give information on the versions, file sizes and generation time (for the document without the logging).

The middle section will give information on timings for each type fo activity. If the trace level is Verbose (or Diagnostic) then the performance metrics will detail specific areas, for eaxmple below we can see that the loading of the google font(s) was causing our template to increase generation time by 110 milliseconds to load the font css. Luckily the font files themselves are cached and did not need to be reloaded each time. But we could save that time by using a local css.

Trace Output

Document Overview

Title	Hello World
Source Template	/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/READMESample.html
Scryber Version	1.0.0.0 (5.0.0.4)
Document Size	371kb 724
Generation Time	120.287ms
Trace Level	Verbose

Performance Metrics

Entry	Duration (ms)	Count
Parse Files	112.79	3
/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/READMESample.html	0.75	
https://fonts.googleapis.com/css2?family=Fraunces:ital,wght@0,400;0,700;1,400;1,700&display=swap	111.74	
/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/Content/HTML/CSS/HelloPrint.css	0.29	
Document Init Stage	0.01	1
Document Load Stage	112.43	1
Document Bind Stage	0.59	1
Document Layout Stage	3.07	1
Document Render Stage	3.31	1
Parse Templates	0.53	7
/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/READMESample.html	0.08	
/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/READMESample.html	0.05	
/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/READMESample.html	0.04	
/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/READMESample.html	0.04	
/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/READMESample.html	0.14	
/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/READMESample.html	0.06	
/Users/richardhewitson/Projects/Scryber.Core/Scryber.Core/Scryber.Core.UnitTest/bin/Debug/net5.0/../../../Content/HTML/READMESample.html	0.12	
Layout Pages	2.92	2
Page sect1	2.89	
Push Component Layout	0.11	1
Text Layout	0.57	22
Text Measure	0.17	22
Style Build	0.53	27
Table Build Process	0.07	4

Entry	Duration (ms)	Count
Image Load	0.06	2

10.26 Namespaces and their Assemblies

Scryber relies on the xml namespaces (xmlns) to identify the classes it should use when parsing an XML or XHTML file. This is based on a mapping of xmlns value to a fully qualified assembly name and namespace.

10.26.1 Declared Namespaces

The 3 base namespaces that are automatically added are:

- **http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Components.xsd**
  - The base level components used in scryber. e.g. Document, TableGrid, List, Span etc.
  - It refers to the Scryber.Components namespace in the Scryber.Components assembly (Version=1.0.0.0, Culture=neutral, PublicKeyToken=872cbeb81db952fe)
- **http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.xsd**
  - The data components used in scryber. e.g. ForEach, Choose, If.
  - It refers to the Scryber.Data namespace in the Scryber.Components assembly (Version=1.0.0.0, Culture=neutral, PublicKeyToken=872cbeb81db952fe)
- **http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Styles.xsd**
  - The style components and attributes used in scryber. e.g. StyleDefn, StyleGroup, StylesDocument.
  - It refers to the Scryber.Styles namespace in the Scryber.Styles assembly (Version=1.0.0.0, Culture=neutral, PublicKeyToken=872cbeb81db952fe)

The html and svg namespaces are also automatically added.

- **http://www.w3.org/1999/xhtml**
  - The html components used in scryber. e.g. div, span, section etc.
  - It refers to the Scryber.Html.Components namespace in the Scryber.Components assembly (Version=1.0.0.0, Culture=neutral, PublicKeyToken=872cbeb81db952fe)
- **http://www.w3.org/2000/svg**
  - The svg drawing components used in scryber. e.g. ellipse, circle, rect etc.
  - It refers to the Scryber.Svg.Components namespace in the Scryber.Components assembly (Version=1.0.0.0, Culture=neutral, PublicKeyToken=872cbeb81db952fe)

Note: If a file or stream of content does not have a namespace, then the classes cannot be found and therefore parsed.

10.26.2 Class attributes

Within the assembly namespaces, referred to above, are actual classes decorated with attributes for each of their properties and contents.

```
namespace Scryber.Html.Components
{
    [PDFParsableComponent("body")]
    public class HTMLBody : Scryber.Components.Section
    {
```

(continues on next page)

(continued from previous page)

```
        [PDFAttribute("class")]
        public override string StyleClass
        {
            get => base.StyleClass;
            set => base.StyleClass = value;
        }

        [PDFAttribute("style")]
        public override Style Style
        {
            get => base.Style;
            set => base.Style = value;
        }

        [PDFElement("")]
        [PDFArray(typeof(Component))]
        public override ComponentList Contents
        {
            get { return base.Contents; }
        }

        [PDFElement("header")]
        [PDFTemplate(IsBlock= true)]
        public override IPDFTemplate Header
        {
            get => base.Header;
            set => base.Header = value;
        }

        [PDFElement("footer")]
        [PDFTemplate(IsBlock = true)]
        public override IPDFTemplate Footer
        {
            get => base.Footer;
            set => base.Footer = value;
        }

        [PDFAttribute("hidden")]
        public string Hidden
        {
            get { return (this.Visible) ? string.Empty : "hidden" }
            set { this.Visible = (string.IsNullOrEmpty(value) || value != "hidden") ?␣
↪true : false; }
        }

        [PDFAttribute("title")]
        public override string OutlineTitle
        {
            get => base.OutlineTitle;
            set => base.OutlineTitle = value;
        }

        public HTMLBody()
            : base()
        {
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Here we can see the html body class decorated with the PDFParsableComponent attribute, so the parser know when it gets to a <body> tag in the content stream in namespace Scryber.Html.Components it create an instance of the HTMLBody class.

The class inherits from the Scryber.Components.Section (an overflowing page), and overrides some of the base functionality to support the standard html attributes. For example the [PDFAttribute("class")] maps the @class attribute in the content stream to the StyleClass string property in the instance.

The explicitly named [PDFElement("head")] if found will be assigned to the Header property, in this case as a template so it can be used multiple times (See *Dynamic content in your template*)

Finally the empty PDFElement attribute with the PDFArray attribute tells the parser it should expect inner child components (that are not nested within another element) of type Component, and they should be added to this collection.

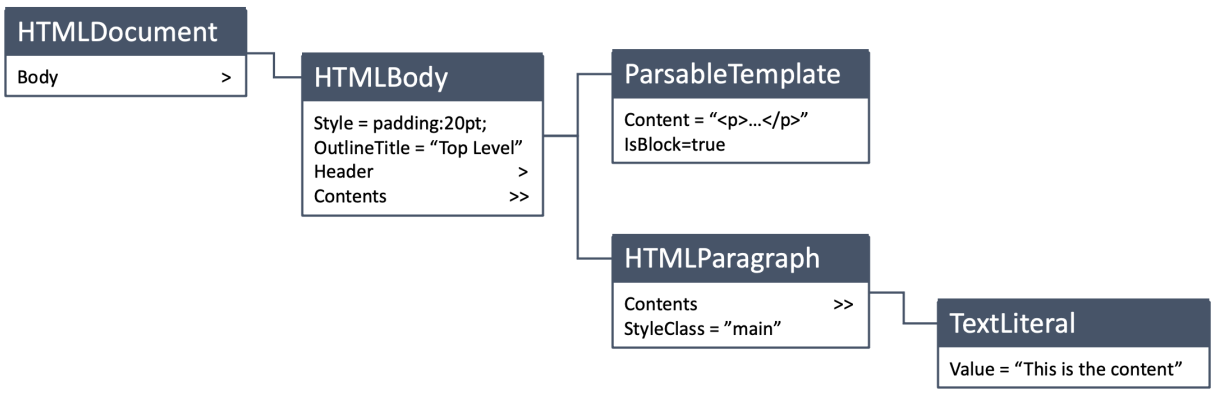
Note: Scryber has an explicit parser, rather than implicit. So if classes or properties are not decorated, then they will not be used.

10.26.3 Parsing the content

Considerring the below content we can see the namespace mapping to the classes

```
<html xmlns='http://www.w3.org/1999/xhtml'>
    <body style='padding:20pt;' title='Top Level' >
        <head><p>This is the header</p><head>
        <p class='main'>This is the content</p>
    </body>
</html>
```

When parsed this will give us an object graph of the below. The content in the header is kept as a string and will be parsed when used each time.



See *Extending Scryber - td* to understand how to add your own classes and namespaces.

10.27 Extending Scryber - td

Content coming soon

10.28 Version History

The following change log is for developers upgrading from previous versions, or looking for new features

10.28.1 Version 5.0.6

Released 30th March 2021

A catch up and fix release for the library, while we are building the docker images and playground.

Minor enhancements and bug fixes

- Support for parsed JSON objects in binding - along with std types and dynamic objects. (See: *Dynamic content in your template*)
- Css 'margin:value' is applied to all margins even if explicit left, right etc. has been previously applied. (See: *Styles in your template*)
- Conformance is now carried through to templates, so errors are not indavertantly raised inside the template. (See: *Scryber Trace Log*)
- Missing background images will not raise an error. (See: *Images in documents*)
- Support for data images (src='data:image/..') within content - thanks Dan Rusu!
- Images are not duplicated within the output for the same source.

10.28.2 Version 5.0.5

Released 28th February 2021

Big Hitters

- Embed and iFrame support. (See: *Splitting into multiple files*)
- Support for border-left, border-right, etc (See: *Colours, fills, borders and backgrounds*)
- Support for encryption and restrictions. (See: *Securing Documents*)
- Support for base href in template files. (See: *File structure*)
- Added em, strong, strike, del, ins elements. (See: *Textual Layout*)

Minor enhancements and bug fixes

- Classes and styles on template tags are supported.
- Html column width and break inside
- CSS and HTML Logging
- Binding speed improvements for longer documents.
- Fixed application of multiple styles with the same word inside
- Allow missing images on the document is now supported.

- Contain fill style for background images.

10.28.3 Version 5.0.4

**Initial SVG Support (See: *Drawing with SVG*) **

Local font urls along with some bug fixes.

10.28.4 Version 5.0.3

- Added @font-face, absolute, relative and display css. (See: *Document fonts and text styles*)
- Support for @page css directives for the whole document and section page sizes. (See: *Document fonts and text styles*)
- Support for <page /> tags with property or for attributes. (See: *Document fonts and text styles*)
- Added support for HTML binding with the template tag and data-bind attribute (See: *Dynamic content in your template*)
- Fix for anchor links with internal and external href.
- Fixes for single character css values and other minor updates.

5.0.1-alpha

** Upgrade to support dotnet 5 **

Plus increased support for the HTML parsing with entities and DTD

10.28.5 Version 1.1 Core Change log

This is a breaking change for existing implementations, but represents a significant step foreward.

- XML content should now use the doc: prefix for the components namepsace
- The Scryber.Components namespace classes no longer have the PDF prefix i.e. PDFDocument is now Document.
- The output of a pdf method has changed SaveAsPDF
- Updated the schemas to match the new document structure

Other changes include the use of the match='[css selector]' on styles with priorities based on depth, and the support for xhtml as a root element in a document parsing.

10.28.6 Version 1.0 Core Change log

The first release of the library for DotNet Core

It includes the switch to a Document/Data element Improved layout capabilities The support for TTC (true type collection fonts) Various other enhancements