Scryber Core

Jan 30, 2021

Getting Started

Scryber.Core is **the** engine to create dynamic documents quickly and easily with consistant styles and easy flowing layout. It's open source; flexible; styles based; data driven and with a low learning curve.

A document generation tool written entirely in C# for dotnet core.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
            xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
            xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd" >
    <Params>
        <pdf:String-Param id="Title" value="Document Title" />
    </Params>

    <Data>
        <data:JsonDataSource id="XmlSource" source-path="http://localhost:5000/Home/
↪Json" ></data:JsonDataSource>
    </Data>

    <Styles>
        <styles:Style applied-type="pdf:H1" applied-class="title" >
            <styles:Background color="#336666"/>
            <styles:Fill color="#FFFFFF"/>
            <styles:Font family="Gill Sans" size="24pt" italic="true"/>
        </styles:Style>
    </Styles>

    <Pages>

        <pdf:Page styles:margins="20pt">
            <Content>
                <data:With datasource-id="JsonSource"  >

                <pdf:H1 styles:class="title" text="{@:Title}" > </pdf:H1>

                <pdf:Ul>
                    <data:ForEach value="{@:.Entries/Entry}" >
                    <Template>
                        <pdf:Li>
                        <pdf:Text value="{@:.Name}" />
                        </pdf:Li>
                    </Template>
                    </data:ForEach>
                </pdf:Ul>
                </data:With>

            </Content>
        </pdf:Page>
    </Pages>

</pdf:Document>
```

CHAPTER 1

Easy, and intuitive structure

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

CHAPTER 2

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 3

Cascading Styles

With a styles based structure, it's easy to apply designs to templates. Use class names, id's or component types, or a combination of all 3 to apply style information to your documents.

CHAPTER 4

Low code, zero code development

Scryber is based around xml templates - just like XHTML. It can be transformed, it can be added to, and it can be dynamic built. By design we minimise errors, reduce effort and allow reuse.

CHAPTER 5

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, or get the document to look up and bind the data for you.

CHAPTER 6

Learn More

Take a look at the quick start guides on Getting started with MVC or Getting started with GUI applications to learn more.

6.1 MVC Controller - Getting Started

A Complete example for creating a hellow world PDF file from an MVC Controller in C#

6.1.1 Nuget Packages

Make sure you install the Nuget Packages

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

This will add the latest version of the Scryber.Core nuget package, and the Scryber.Core.Mvc extension methods.

Add the XML Schema files to help with the intellisense

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

6.1.2 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 a create a new XML file called HelloWorld.pdfx in your folder.

And paste the following content into the file

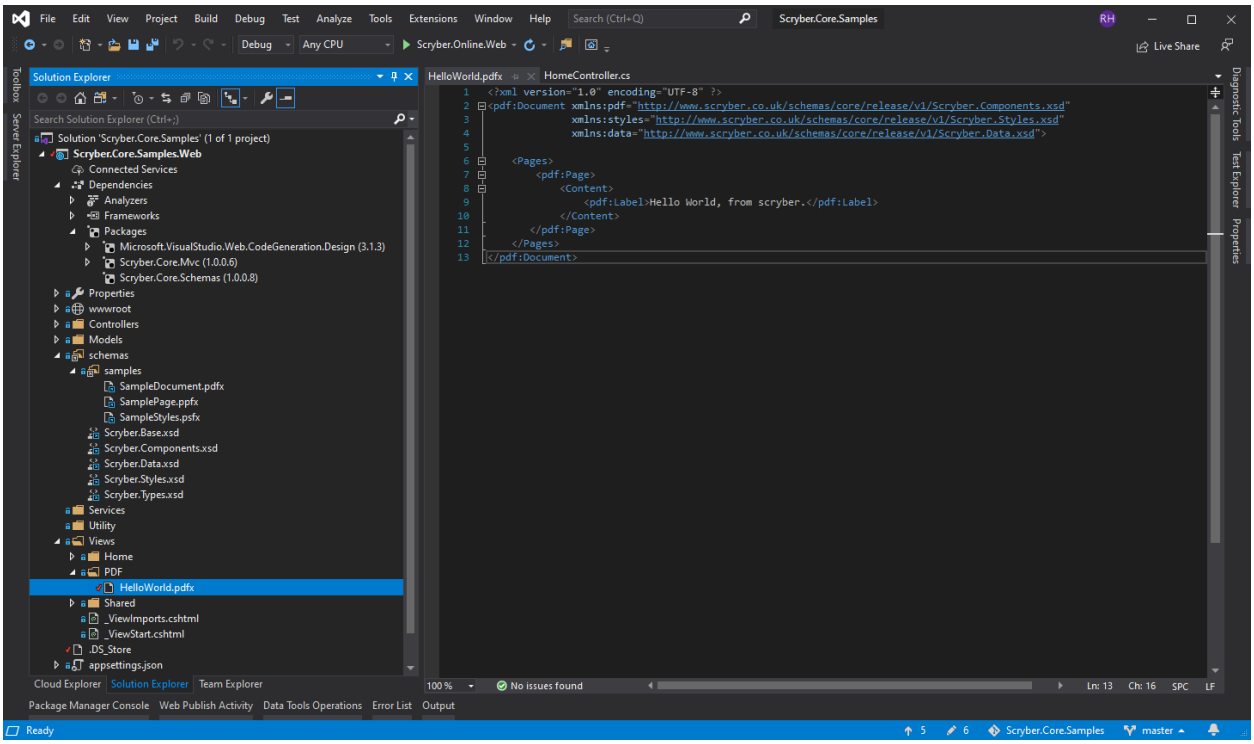
```
<?xml version="1.0" encoding="UTF-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
          xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
```

(continues on next page)

(continued from previous page)

```
        xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd">
    <Pages>
        <pdf:Page>
            <Content>
                <pdf:Label>Hello World, from scryber.</pdf:Label>
            </Content>
    </pdf:Page>
    </Pages>
</pdf:Document>
```

Your solution should look something like this.



For for more information on the namespaces and mappings see this [About Namespaces](namespaces-and-assemblies) documentation

6.1.3 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;
```

6.1.4 Add the Web host service

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

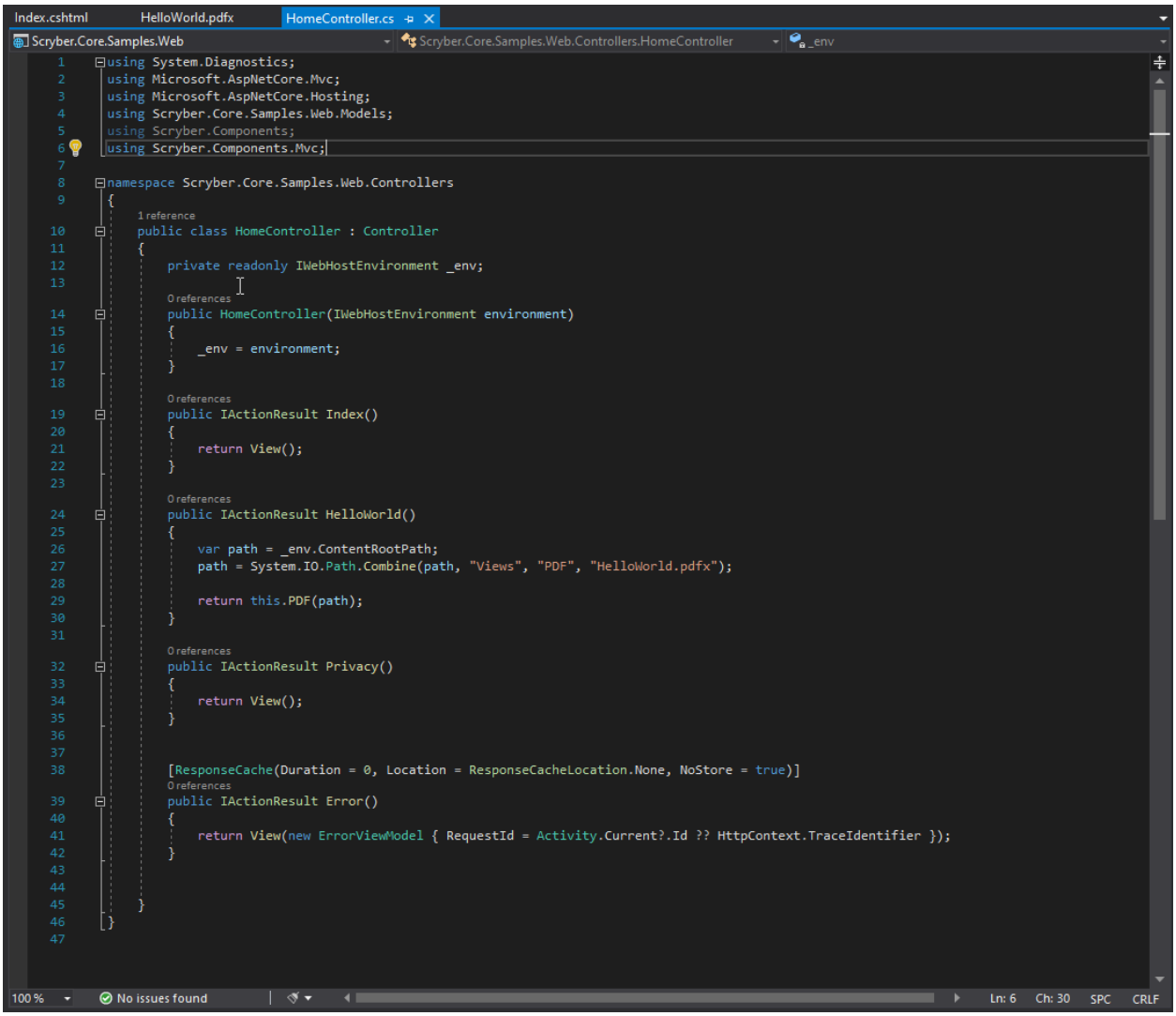
```
private readonly IWebHostEnvironment _env;

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

6.1.5 Add a Controller Method

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

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

    return this.PDF(path);
}
```

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

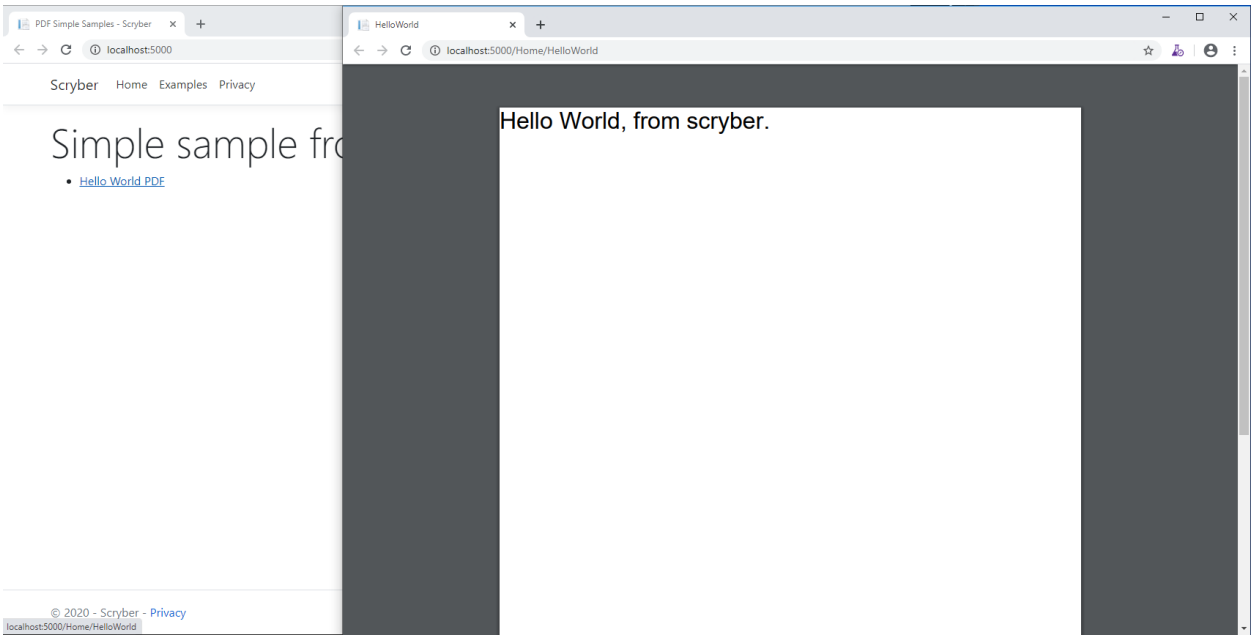


6.1.6 Testing your action

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>
    <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.



6.1.7 Further reading

You can read more about what you can do with scryber here

- *Passing data to your template*
- *A Scryber XML File structure*
- component_types
- *Styles in your template*
- *Splitting into multiple files*

6.2 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.

6.2.1 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.

Add the XML Schema files if you want to have help with the intellisense on the XML files

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

6.2.2 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 a create a new XML file called HelloWorld.pdfx in your folder.

And paste the following content into the file

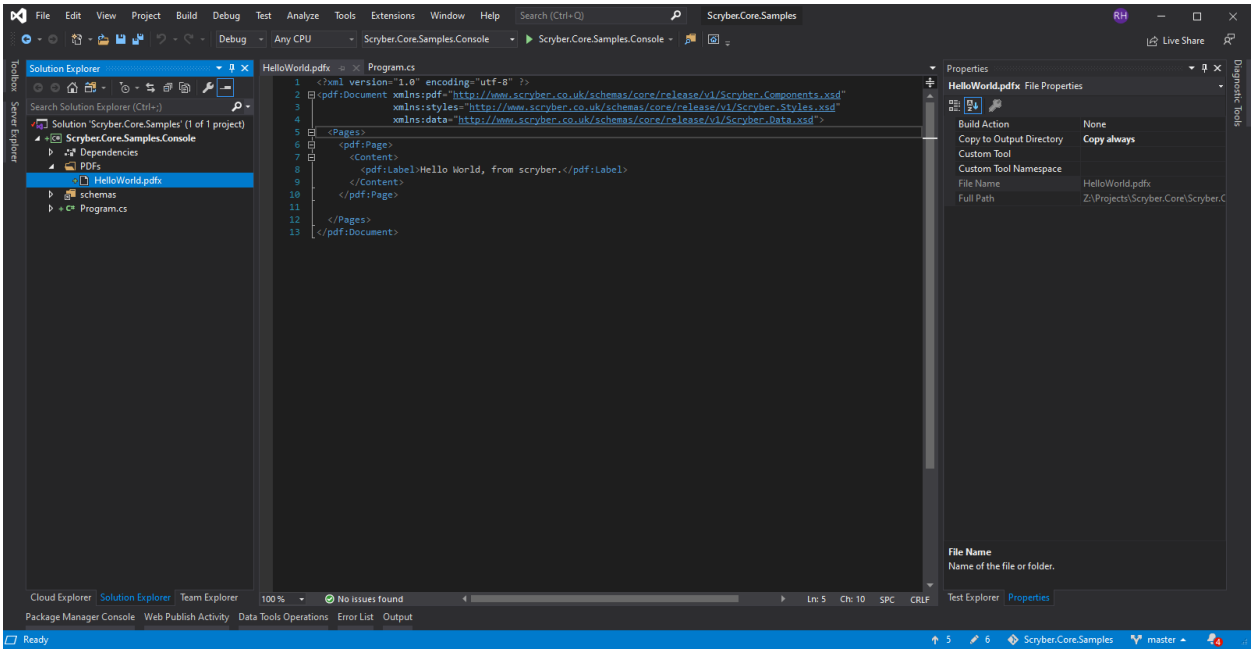
```
<?xml version="1.0" encoding="UTF-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
          xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
          xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd">
    <Pages>
        <pdf:Page>
            <Content>
                <pdf:Label>Hello World, from scryber.</pdf:Label>
            </Content>
        </pdf:Page>
    </Pages>
</pdf:Document>
```

For for more information on the namespaces and mappings see this [About Namespaces](namespaces_and_assemblies) documentation

Pdfx file properties

In the file properties for the HelloWorld.pdfx 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.



6.2.3 Program code

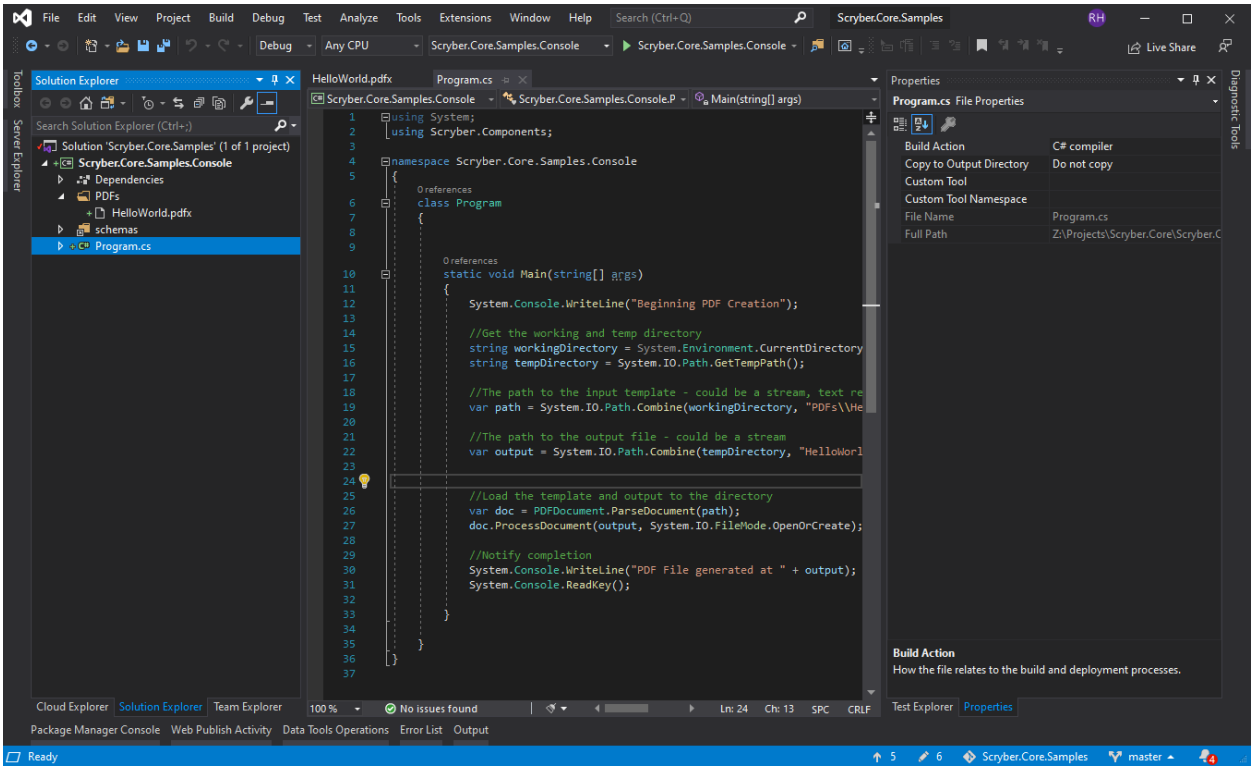
In your program.cs add the namespace to the top of your class.

```
using Scryber.Components;
```

6.2.4 Replace your main program method.

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

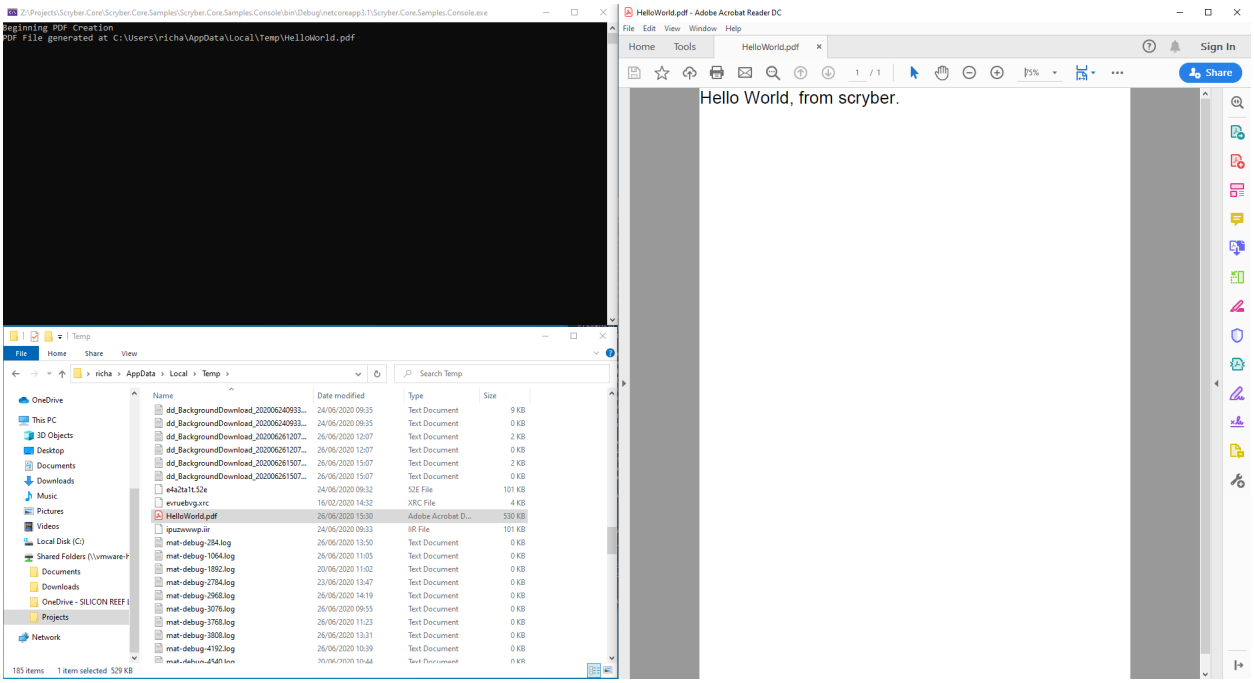
```
static void Main(string[] args)
{
    System.Console.WriteLine("Beginning PDF Creation");

    //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.pdfx");

    //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 = PDFDocument.ParseDocument(path);
    doc.ProcessDocument(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.

6.2.5 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.



6.2.6 Further reading

You can read more about the what you can do with scryber here:

- *Passing data to your template*
- *A Scryber XML File structure*
- component_types
- *Styles in your template*
- *Splitting into multiple files*

6.3 To code or not to code...

Scryber does not rely on xml, but it makes life easier and is more visual and structured. But it does hand in hand you code. When ever you parse a PDFDocument or component you are simply creating the same as you could in code.

6.3.1 XML Template

```
<?xml version="1.0" encoding="UTF-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
          xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
          xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd">
    <Pages>
        <pdf:Page>
            <Content>
                <pdf:Label>Hello World, from scryber.</pdf:Label>
            </Content>
        </pdf:Page>
    </Pages>
</pdf:Document>
```

6.3.2 The same in code

```
var doc = new PDFDocument();
var page = new PDFPage();
doc.Pages.Add(page);

var label = new PDFLabel();
label.Text = "Hello World, from scryber";
page.Contents.Add(label);

return doc.ProcessDocument();
```

6.3.3 Loading from a stream

It is always possible to mix the declarative with the code. For example we can load the XML as a string or stream and inject the data as needed.

```
public IActionResult DocumentDynamic(string title = "New Document")
{
    //Load the content and model in an MVC Controller method
    using (var pdfx = GetDocument(title))
    {
        var model = GetData(title);

        //And output the content as together
        return this.PDF(pdfx, model);
    }
}

protected PDFDocument GetDocument(string title)
{
    string content = @"<?xml version='1.0' encoding='utf-8' ?>
                <pdf:Document xmlns:pdf = 'http://www.scryber.co.uk/schemas/core/
↪release/v1/Scryber.Components.xsd'
                              xmlns:styles = 'http://www.scryber.co.uk/schemas/core/
↪release/v1/Scryber.Styles.xsd'
                              xmlns:data = 'http://www.scryber.co.uk/schemas/core/
↪release/v1/Scryber.Data.xsd' >
                <Params>
                    <pdf:Object-Param id='Model' ></pdf:Object-Param>
                </Params>
                <Pages>
                    <pdf:Section>
                        <Content>
                            <data:ForEach id='Foreach2' value='{@:Model.Entries}' >
                                <Template>
                                    <pdf:Label text='{@:.Name}' /><pdf:Br/>
                                </Template>
                            </data:ForEach>
                        </Content>
                        <Footer>
                            <pdf:Div styles:padding='5pt' styles:h-align='Center' >
                                <pdf:PlaceHolder contents='{@:Model.Footer}' />
                            </pdf:Div>
                    </pdf:Section>
                </Pages>
            </pdf:Document>";

    //With a string reader, but could be any stream, text reader, xml reader or other␣
↪source.
    using (var reader = new System.IO.StringReader(content))
    {
        return PDFDocument.ParseDocument(reader, ParseSourceType.DynamicContent);
    }
}

protected object GetData(string title)
{
    var data = new
    {
        Title = title,
        Entries = new[]
            {
                new { Name = "First", Id = "FirstID"},
```

(continues on next page)

(continued from previous page)

```
                new { Name = "Second", Id = "SecondID"}
            },
        Footer = "<pdf:PageNumber />"
    };
    return data;
}
```

Note: When loading from a stream, there is no relative reference to a local file. If you need to reference other files do so relative to the working directory, or pass in your own IPDFReferenceResolver

6.3.4 Why use one over the other

We always think that the declarative is better for what you need, but sometimes building in code works. Using the *Using controllers with your templates - td* allows you to hook content back into a document template.

In this documentation, we will concentrate on the use of the declarative XML with code where appropriate, but remember that everything that is declared can be coded too.

6.4 A Scryber XML File structure

At the root, the Document file (.pdfx) has a number of capabilities that change the output content.

- Scryber Processing Instruction
- Namespaces - The namespaces used in the document.
- Params - The parameters or variables in a document.
- Data - Contains any referenced data sources in a document.
- Styles - Contains and styles (in document or referenced).
- Pages - The visual page content in the document or referenced.
- Viewer Options - Options for how the document opens in the reader.
- Render Options - Options for altering how the pdf is output.
- Info - Metadata options about the document itself.

6.4.1 Example

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

<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
            xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
            xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd" >

    <Viewer hide-toolbar="true" page-layout="TwoColumnRight" center-window="true" />
```

(continues on next page)

(continued from previous page)

```
    <Render-Options ></Render-Options>
    <Info></Info>

    <Styles>
    </Styles>

    <Params>
        <pdf:String-Param id="MyTitle" value="Document Title" />
    </Params>

    <Pages>
        <pdf:Page outline-title="Third Page">
        <Content>
            <pdf:H1 styles:class="title" text="{@:MyTitle}" > </pdf:H1>
        </Content>
        </pdf:Page>
    </Pages>

</pdf:Document>
```

Note: The order of the elements is not important, but it's a maximum of one each. It is not an error for a Document element to be completely empty, but it will be just that.

6.4.2 Processing Instructions

The scryber processing instruction is an optional entry at the very top of the xml file or content stream to define explicit options for the way the file is handled.

```
<?xml version="1.0" encoding="utf-8" ?>
<?scryber append-log='true' log-level='Messages' parser-log='true' ?>
<pdf:Document ....
```

As a processing instruction, the schema xsds do not support validation of the content, but the following are the supported options.

- **'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 - Be carefull, 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.
- **'append-log' - Controls the tracing log output for a single document (see document_logging)**
  - false - This is the default and the document will be rendered and output 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.
- **'parser-culture' - specifies the culture settings when parsing a file for interpreting dates and number formats in the xml. e**
  - 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 - 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).
- **'controller' - This is the full type name of a controller for the docucment, that can interact with and handle events on the d**
  - 'Namespace.TypeName, AssemblyName' - The class should have a parameterless constructor (see: *Using controllers with your templates - td*)

6.4.3 Namespaces

Scryber is dynamic and extensible. The xml namespaces refer directly to namespaces (and assemblies) in the library. There are 3 primary namespaces, and a convention for the prefixes for those namespaces.

By using explicit namespaces the xml can be read and which class an element rerers to determined. Scryber requires the use of a prefix for all of the namespaces (as there are content elements defined without a prefix).

- ***pdf* - http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Components.xsd**
  - These are the main visual and structural components in a file or document.
  - e.g. pdf:Document; pdf:Page; pdf:Label.
  - It refers to the assembly namespace *Scryber.Components, Scryber.Components, Version=1.0.0.0, Culture=neutral, PublicKeyToken=872cbeb81db952fe*
  - see *Standard document components* for more on teh content elements of a document.
- ***data* - http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Styles.xsd**
  - These are either non visual components that load data from other sources, change content based on rules in data sources, or create inner content based on available data.
  - e.g. data:DataGrid; data:XmlDataSource; data:If.
  - It refers to the assembly namespace *Scryber.Data, Scryber.Components, Version=1.0.0.0, Culture=neutral, PublicKeyToken=872cbeb81db952fe*
  - see *Data Binding in Scryber - td* for more on working with data.

- ***styles*** **- http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.xsd**
  - These apply colour, size and other visual style to the components, both as Style elements and as attributes on pdf:Compontents
  - e.g. styles:Style; styles:bg-color; styles:width.
  - It refers to the assembly namespace *Scryber.Styles, Scryber.Styles, Version=1.0.0.0, Culture=neutral, PublicKeyToken=872cbeb81db952fe*
  - see *Styles in your template* for more on working with styles.

For more information on how these are mapped, and also adding your own namespaces see *Namespaces and their Assemblies - td* along with <*no title*>

6.4.4 Params

The 'Params' element (short for parameters) contains the strongly typed values for parameters (aka variables) that can be used in the document, to alter the final content output. They form a key part of the document creation process, and allow creators to pass information from their code into the template(s).

In fact they are so key to PDF generation, they have their own section, terminology, and functions (see: *Parameters and object binding*).

6.4.5 Data

The *Data* element is a new element for version 1.0. Previously all sources of data would be mixed into the content of the document. With the separation and full support for data binding (see: *Data Binding in Scryber - td*), with a top level element it's easier to structure and separate and keep the visual content in Pages.

Note: It's not an error to put your data components in the Pages section, as there can be times when it's really useful.

6.4.6 Styles

The *Styles* element contains all the document style class information that is not inline of the components themselves, along with any references to stylesheets. Document styles (*Styles in your template*) are fully bindable to any data components or prameters.

6.4.7 Pages

The *Pages* element contains the visual content of the document. Whether that is single pages, sections of multiple pages, or references to external pages. See *Pages and Sections* for more information on the visual content.

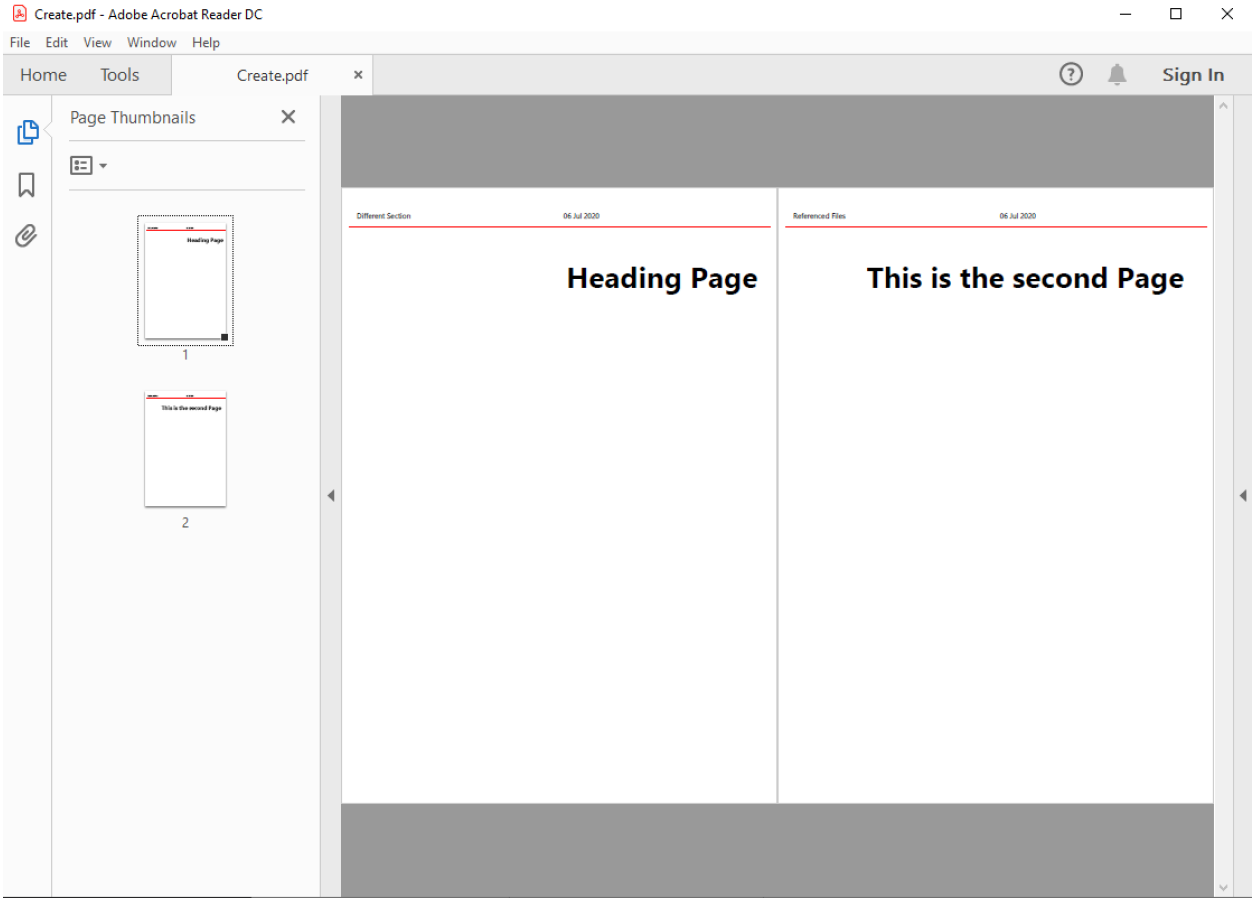
6.4.8 Viewer Options

The viewer options within the Document level element alter how readers (should) show the document and it's contents. Not all readers support these (especially browsers), but it can help.

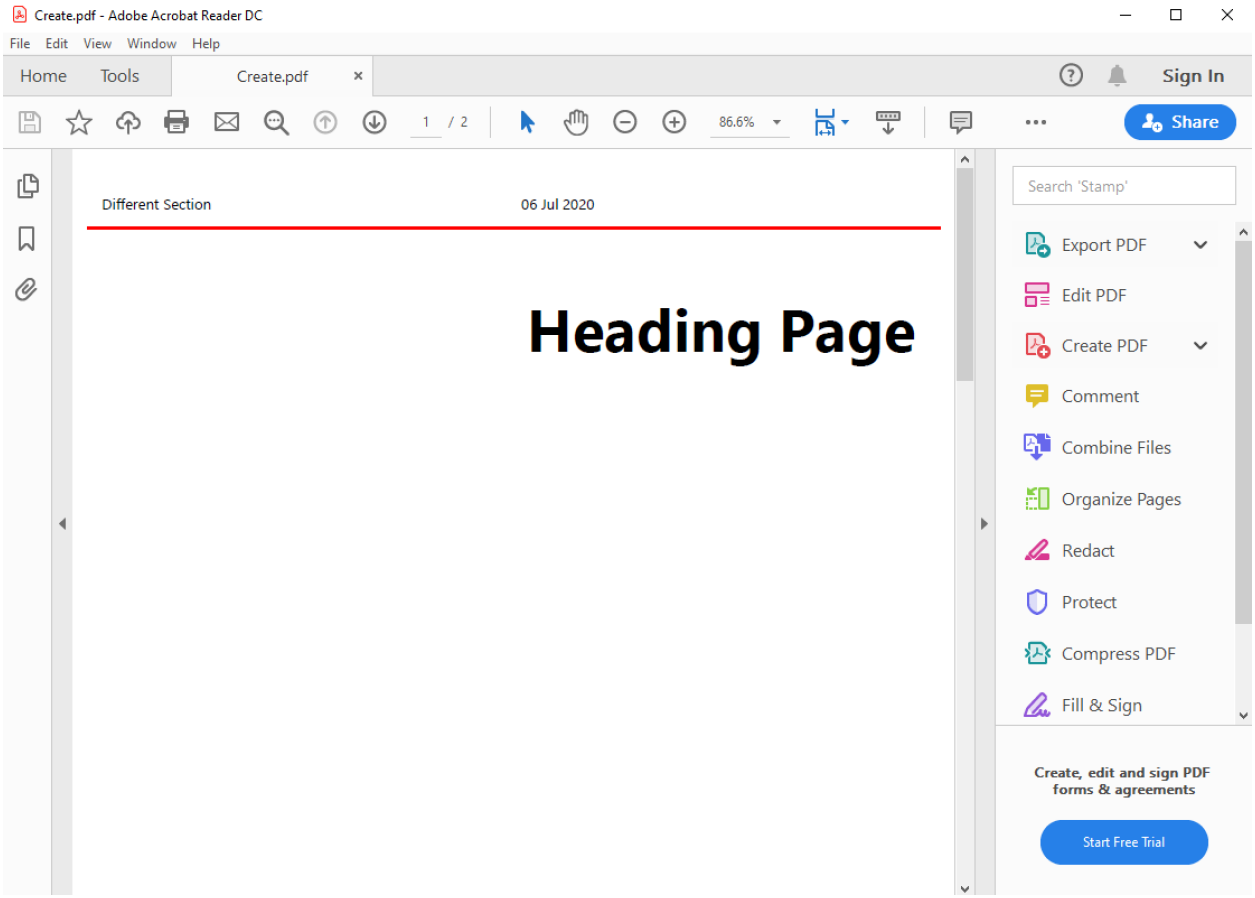
For example the following viewer options:

```
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
          xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
          xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd"
          auto-bind="true" >
<Viewer hide-toolbar="true" page-display="Thumbnails" page-layout="TwoPageLeft" fit-
↪window="false" />
```

Will open in Acrobat Reader as:



Whereas without the View options the default is:



The following options are declared and supported in the Viewer element

- *hide-toolbar* - will show or hide the toolbar (currently a side bar) in reader.
- ***page-display* - Indicates the type of side navigation shown for the document. Supported values are:**
  - *None* - Side display is hidden (contracted).
  - *Thumbnails* - The page thumbnails are shown.
  - *Outlines* - The document outline, a hierarcial structure of the content, is shown. (see *A Scryber Document outline - td*)
  - *Attachments* - The document attachments panel is shown.
  - *FullScreen* - This attempts to open the document in full screen presentation mode. A warning to the end user is often shown beforehand.
- ***page-layout* - Indicates how pages will be displayed in the view. Supported values are:**
  - *SinglePage* - It will open with a page per view sizing in the reader window.
  - *TwoPageLeft* - The document will open with a side by side view of 2 pages, where the first page is on the left.
  - *TwoPageRight* - The document will open with a single first page (the right page) and then 2 page per view following that. Very similar to reading a book.
  - *OneColumn* - The document will open with a full width continuous display, to support scrolling through the complete document.
  - *TwoColumnLeft* - 2 pages, side by side with a full width continuous display.

  - *TwoColumnRight* - 2 pages, side by side, continuous scrolling, with the first page on it's own as per a book.

- *fit-window* - If true the window will resize to fit the width of the first page.
- *center-window* - If true, the UI reader window will center in the main screen.
- *hide-menubar* - If true, then the window menu bar should be hidden.

6.4.9 Render Options

This element controls the output of the PDF content itself. Most of the default values are correct for the best output, but can be altered if needed.

Note: This section is primarily so the contents of the output PDF can be inspected and read, looking at PDF contents is not recommended for anyone with a sensitive bladder or prone to feinting.

```
<Render-Options component-names="ExplicitOnly" compression-type="None" output-
↪compliance="None"
img-cache-mins="10" string-output="Hex" ></Render-Options>
```

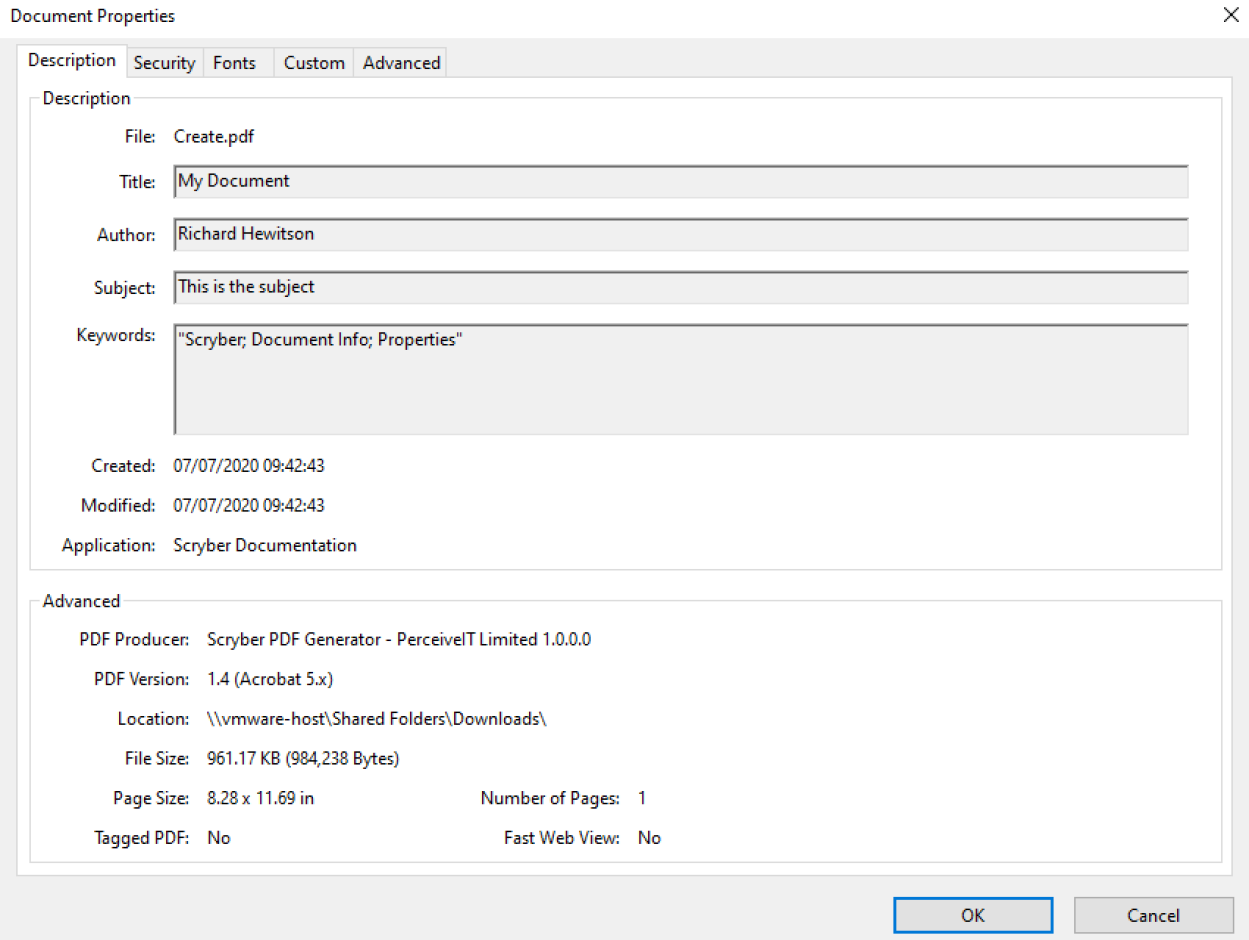
The following options are supported on the render options element.

- ***component-names* - Defines how the output of names will be used. This is important for linking to sections from within the**
  - *ExplicitOnly* - (default) Only the components that have an actual name value will be listed.
  - *All* - Any component with an ID will be included in the name dictionary, and can be linked to with the UniqueID (full path with underscores).
- ***compression-type* - Defines whether the indirect streams within a pdf are compressed or as plain text.**
  - *FlateDecode* - (default) The stream content will be zip compressed to reduce the file size.
  - *None* - The streams will be put in the file in their raw format. File size will increase, but the streams can be 'read'
- ***string-output* - Defines how strings of text within the document and object streams are written to the file.**
  - *Hex* - (default) The textual values will be written as Hex encoded values. This is better for unicode characters.
  - *Text* - The string characters will be output with the ACSII format encoding of PDF files, and unicode will be escaped but render badly.
- ***img-cache-mins* - Defines within the document any images that are used will be cached for re-use, and not loaded from a fi**
  - *0* - (default) The images will be loaded each time for a document creation
  - *60* - Specify any number of minutes to hold the images in the cache. Changes to the images will not be updated in the documents for that time either.

The Render element also allows an inner <*Writer*> element. This can support other writers including the secure writer and the protected writer. (coming soon)

6.4.10 Document Info

This controlls the output document information, that can be seen in the document properties of acrobat reader. This information is also, open and indexed by many search engines and forms the key metadata about the document.

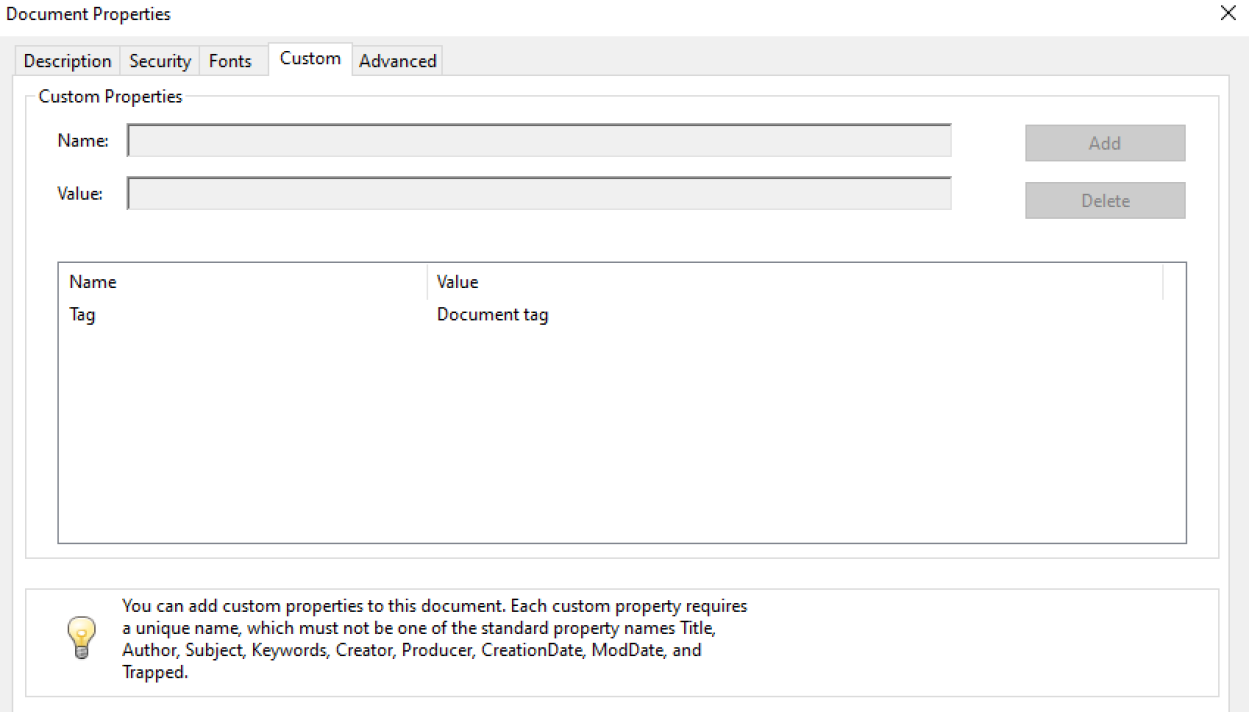


It is also bindable on the main attributes and named elements so can be changed at runtime. (see '*Parameters and object binding*' for more on the parameters and binding).

```
<Info creator="Scryber Documentation"  >
    <Title>{@:Title}</Title>
    <Subject>{@:Subject}</Subject>
    <Keywords>{@:Keywords}</Keywords>
    <Author>{@:Author}</Author>
    <pdf:Extra name="Tag" >Document tag</pdf:Extra>
</Info>

<Params>
    <pdf:String-Param id="Title" value="Document Title" />
    <pdf:String-Param id="Author" value="My Name" />
    <pdf:String-Param id="Subject" value="This is the subject" />
    <pdf:String-Param id="Keywords" value="Scryber; Document Info; Properties" />
</Params>
```

The attributes on the document *Info* for creator, created-date, producer, modified-date are automatically completed by the scryber library but can be overriden.

The pdf:Extra entries within the info, allow producers to add their own keywords and content. This will appear in the custom tag of the document properties, but can also be processed by search engines and other tools quickly and easily.



6.5 Standard document components

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

6.5.1 Document level visual components

- **Page**
  - A single page, where content that will extend beyond the boundaries is truncated
  - Has an optional page header and page footer, as well as content.
  - see *Scryber.Components.PDFPage*
- **Section**
  - A set of pages of the same size and orientation, where content that flow onto the next page.
  - Has an optional continuation page header and footer, along with the page header, page footer and content.
  - see reference/pdf_section
- **Page-Group**
  - A group of pages that can have shared size, header and footer content, and style.
  - Individual pages can override as needed.
  - see reference/pdf_pagegroup

- **Page-Ref**
  - A reference to one of the above components in a separate file.
  - Specified via a required source attribute.
  - see *Splitting into multiple files* and *Scryber.Components.PDFPage*

6.5.2 Standard structural components

- **Div**
  - a block level component that will fill the width of the available parent.
  - see *Scryber.Components.PDFDiv*
- **Span**
  - an inline compnent that can have any content including text.
  - see reference/pdf_span
- **Table**
  - A grid of rows and cells (that can be spanned across columns.
  - see *Scryber.Components.PDFTable*
- **Lists**
  - Ordered lists with numbering styles.
  - Unodered lists with a bullet styles.
  - Definition lists with a label and content.
  - see *Scryber.Components.PDFList*
- **Paragraph**
  - A textual (and other content), that has a more defined style than a div.
  - see reference/pdf_para
- **Heading (1 to 6)**
  - A textual (and other content), that is given a pre-defined style based on it's level of 1 to 6
  - H1, H2, H3, H4, H5, H6
  - see reference/pdf_headings
- **Layer-Group**
  - A wrapper for a set of Layers.
  - Each layer will be relatively positioned (default to 0,0) ontop ove each other.
  - Layers can be shown and hidden as needed.
  - see reference/pdf_layergroup
- **Canvas**
  - A drawing panel that will by default relatively position all child components
  - see reference/pdf_canvas
- **Block Quote**

  - A panel with specific margins.
  - see reference/pdf_blockquote

- **Preformatted**
  - A container for pre-formatted text, that will not flow over new lines, or remove line breaks (by detault).
  - see reference/pdf_pre

- **Component-Ref**
  - A reference to an external file or stream that will be injected into the page at runtime.
  - see *Splitting into multiple files*

- **Column break**
  - Stops the flow of content within the current region, and moves any following content onto the next available column.
  - Can be positioned at any depth within a multicolumn layout.
  - see document_layout and reference/pdf_columnbreak

- **Page break**
  - Stops the flow of content within the current page, and moves any following content onto the next available page.
  - Can be positioned at any depth within a layout.
  - see document_layout and reference/pdf_pagebreak

6.5.3 Textual components

- **Text**
  - A text literal compenent where the text can be set to the @value attribute.
  - Supports full data binding.
  - see reference/pdf_text

- **Number**
  - A litteral component that supports numeric values (@value attribute as well as number formatting (@styles:number-format)
  - Can display numbers in any of the standard floating point, currency and integral types.
  - see reference/pdf_number

- **Date**
  - A litteral component that supports date time values (@value attribute as well as date formatting (@styles:date-format)
  - Can display dates in any of the standard localized formats.
  - see reference/pdf_date

- **Label**
  - A text literal component where the text can be set to the @text attribute.
  - Supports full data binding.

  - The only difference is a more formal distinction of purpose than text.
  - see reference/pdf_label

- **PageNumber**
  - A textual component that displays the current output page number where the component is placed.
  - Supports the use of page section counting and total document page count.
  - see reference/pdf_pagenumber
- **PageOf**
  - A textual compenent that displays the page number of a referenced component.
  - Supports the use of page section counting and total document page count.
  - see reference/pdf_pageof
- **Link**
  - A hyper link to a location within the current document, or another document, or a web resource.
  - Content within can be styled appropriately.
  - Document references can be based on ID or name.
  - Page links can be First, Previous, Next, Last or numbered.
  - see document_linking and reference/pdf_link

6.5.4 Graphical components

- **Images**
  - A static or dynamic image loaded from a source, and inserting 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.
  - see *Images in documents - td* and reference/pdf_image
- **Horizontal Rule**
  - A single line within the flow of the document.
  - Can be styled as a independant component.
  - see reference/pdf_hr
- **Line, Rect, Polygon, Ellipse, Path**
  - Standard drawing components that can be used either within the flow of the content or for drawing/designs.
  - see *Drawing paths and shapes - td*
  - and for individual components reference/pdf_line, reference/pdf_rect, reference/pdf_ellipse, reference/pdf_polygon, reference/pdf_path

6.5.5 Data visual components

For a general use of the data components see *Passing data to your template* and *Data Binding in Scryber - td*. And for an overview of the data sources available see document_datasources

- **ForEach**
  - Loops through each value in a data source, with an optional step, offset and count.
  - Outputs the content within the tempate, that can be any inner content.
  - see *Scryber.Data.PDFForEach*
- **DataGrid**
  - Loops through each value in a data source.
  - Outputs the content as a table of results, with various column types.
  - Allows for auto population from a schema in a data source.
  - Also supports alternating styles, fotters and headers.
  - see *Scryber.Data.PDFDataGrid*
- **DataList**
  - Loops through each value in a data source, with an optional step, offset and count.
  - Outputs the content as panels, lists, or spans.
  - Allows for auto population from a schema in a data source.
  - Also supports output order, flow direction, and alternating styles.
  - see *Scryber.Components.PDFDataList*
- **With**
  - Takes a data value or source and applies it to the current context so it can be used in binding statements.
  - Can have any content, and they are full components, rather than templates.
  - Supports both xml and object values.
  - see *Scryber.Data.PDFWith*
- **WithFieldSet**
  - Takes a data value or source and applies it to the current context so it can be used in binding statements.
  - Supports the use of fields within the block to automatically create the content.
  - Allows for auto population from a schema in a data source.
  - Supports both xml and object values.
  - see reference/data_withfieldset
- **Choose**
  - Optionally displays a set of content based on a decision (test).
  - Allows multiple reference/data_ChooseWhen to be defined within the component.
  - The first true decision will be output, and all others not rendered in the document.
  - Allows the use of one reference/data_ChooseOtherwise component as a catch all.
  - see reference/data_choose

- **If**
  - Optionally displays a set of content based on a decision (test).
  - If the decision is false, then no inner content will be rendered.
  - see reference/data_if

6.5.6 Html components

- **Html Page**
  - A full section that supports the inclusion for html (or markdown) content output within a document as it's own page(s).
  - Supports the use of inline style conversion (with limitations) to scryber styles.
  - Content can either be loaded dynamically by the component, assigned from a data source, or explicitly set from code.
  - see using_html for more information on Html in scryber.
  - see reference/pdf_html
- **Html Fragment.**
  - A block of html that can sit within a document.
  - Supports the use of inline style conversion (with limitations) to scryber styles.
  - Content can either be loaded dynamically by the component, assigned from a data source, or explicitly set from code.
  - see using_html for more information on Html in scryber.
  - see reference/pdf_htmlfragment

6.6 Styles in your template

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

6.6.1 Styles on components

Styles are supported on each component within the template. They are based on the styles namespace xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Styles.xsd".

```
<pdf:Div styles:margins="20pt" styles:padding="4pt" styles:bg-color="#FF0000"
            styles:fill-color="#FFFFFF" styles:font-family="Arial" styles:font-size=
↪"20pt" >
    <pdf:Label>Hello World, from scryber.</pdf:Label>
</pdf:Div>
```

Or in the code

```
private static PDFComponent StyledComponent()
{
    var div = new PDFDiv()
    {
        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 PDFLabel()
    {
        Text = "Hello World from scryber"
    });

    return div;

}
```

6.6.2 Style Classes

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

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
            xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
            xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd">
    <Styles>

        <styles:Style *applied-class="mystyle"* >
            <styles:Margins all="20pt"/>
            <styles:Padding all="20pt"/>
            <styles:Font family="Arial" size="20pt"/>
            <styles:Background color="#FF0000"/>
            <styles:Fill color="white"/>
        </styles:Style>

    </Styles>
    <Pages>
        <pdf:Page>
            <Content>

                <pdf:Div styles:class="mystyle">
                    <pdf:Label>Hello World, from scryber.</pdf:Label>
                </pdf:Div>

            </Content>
        </pdf:Page>
```

(continues on next page)

(continued from previous page)

```
    </Pages>
</pdf:Document>
```

By using styles, it's the same as html and css. 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 referenced stylesheet

6.6.3 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.

6.6.4 Applying Styles

Styles can be applied to an element based upon a combination of 3 attributes of the Style.

@applied-id @applied-class @applied-type

e.g.

```
<Styles>

    <!-- 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. -->

    <styles:Style applied-type="pdf:Document" >
        <style:Font family="Gill Sans" size="14pt" />
        <style:Fill color="#333" />
    </styles:Style>

    <!-- This style will be applied to all top level headings
         specifying the font size and some spacing -->

    <styles:Style applied-type="pdf:H1" >
        <styles:Font bold="true" size="30pt" />
        <styles:Margins top="20pt" />
        <styles:Padding all="5pt" />
    </styles:Style>

    <!-- 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.  -->

    <styles:Style applied-class="warning">
```

(continues on next page)

(continued from previous page)

```
        <styles:Background color="#FF0000"/>
        <styles:Fill color="#FFFFFF" />
    </styles:Style>

    <!-- This style will be applied to all components with a class of 'border'
         and give a background colour of red with white text -->

    <styles:Style applied-class="border">
        <styles:Border color="#7777" width="1pt" style="Solid"/>
        <styles:Fill color="#444" />
    </styles:Style>

    <!-- This style will be applied to all H1 Headings with a class of 'border'
         and give a border colour of red with white text -->

    <styles:Style applied-type="pdf:H1" applied-class="border">
        <styles:Border color="#550000" />
        <styles:Fill color="#550000" />
    </styles:Style>

    <!-- This style will only be applied to a component with ID 'FirstHead'
         and give a font size of 48pt -->

    <styles:Style applied-id="FirstHead">
        <styles:Font size="48pt"/>
    </styles:Style>

</Styles>
```

Note: Currently scryber does not support the concept of nested or path styles as css e.g. div.class -> h1.class. It may be supported in the future.

6.6.5 The Applied Type

A style definition with an applied-type attribute is used on all components of that type. It also supports inheritance. The format is based on the qualified component name.

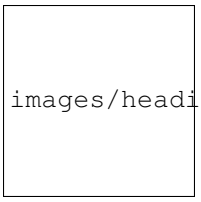
A sample set is give in the schema intellisense if used, but as long as the type name is known the it can be used. Even invisible components such as data:ForEach can have styles applied.

6.6.6 Applying Multiple Styles

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

```
<pdf:H1 id="FirstHead" styles:class="warning border" styles:font-italic="true" >This␣
↪is the Warning heading</pdf:H1>
```

This will apply the H1 style, the 2 classes for the warning and border and increase the size based on the ID of first head. And then the inline italic style will be applied.



6.6.7 Late binding 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.psfx" : "Sheet2.psfx"

var doc = PDFDocument.ParseDocument("MyPath.pdfx");

//Load the stylesheet as a referenced component
var styles = PDFComponent.Parse(sheet) as Styles.PDFStylesDocument;

//and add it to the document styles.
doc.Styles.Add(styles);
```

6.6.8 Data binding Styles

The process of data-binding (see: *Lifecycle of a document creation - td*, and *Data Binding in Scryber - td*) can apply values to styles (and the referenced styles).

e.g.

```
<Params>
    <pdf:Color-Param id='theme-bg' value='#FFFFFF'/>
    <pdf:Color-Param id='theme-bg2 value='#AAAAAA'/>
    <pdf:Color-Param id='theme-title-font' value='Helvetica'>
</Params>
<Styles>

    <!-- 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. -->

    <styles:Style applied-class="title" >
        <style:Font family="{@:theme-title-font}" size="14pt" />
        <style:Background color="{@:theme-by2}" />
    </styles:Style>

</Styles>
```

Here the font family and background for any component with the class title assigned, will pick up the default theme values. Were the code can override these values and provide new colours and fonts for output.

```
var doc = PDFDocument.ParseDocument(path);
doc.Params["theme-bg"] = new Scryber.Drawing.PDFColor(0.0, 0.0, 0.0);
doc.Params["theme-bg2"] = new Scryber.Drawing.PDFColor(0.3, 0.3, 0.3);
```

(continues on next page)

(continued from previous page)

```
doc.Params["theme-title-fill"] = new Scryber.Drawing.PDFColor(1, 1, 1);
doc.Params["theme-title-font"] = "Gill Sans";

return this.PDF(doc);
```

As per object databinding, you can even provide a specific class for binding.

6.6.9 Order and Precedence

Scryber has a very basic precedence order - based on the order in the document.

1. The style from the parent is collected.
2. Any styles in the document are evaluated in the order they appear.
3. If a stylesheet reference is encountered, then the styles within it will be evaluated before moving on to the following siblings
4. Finally the styles directly applied will be evaluated, giving the final result.

This will then be flattened and used in the layout and rendering of the component.

6.7 Positioning 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.

6.7.1 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.

Carriager returns within the content of the xml file are ignored by default, as per html (see reference/pdf_pre if you don't want them to be.).

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

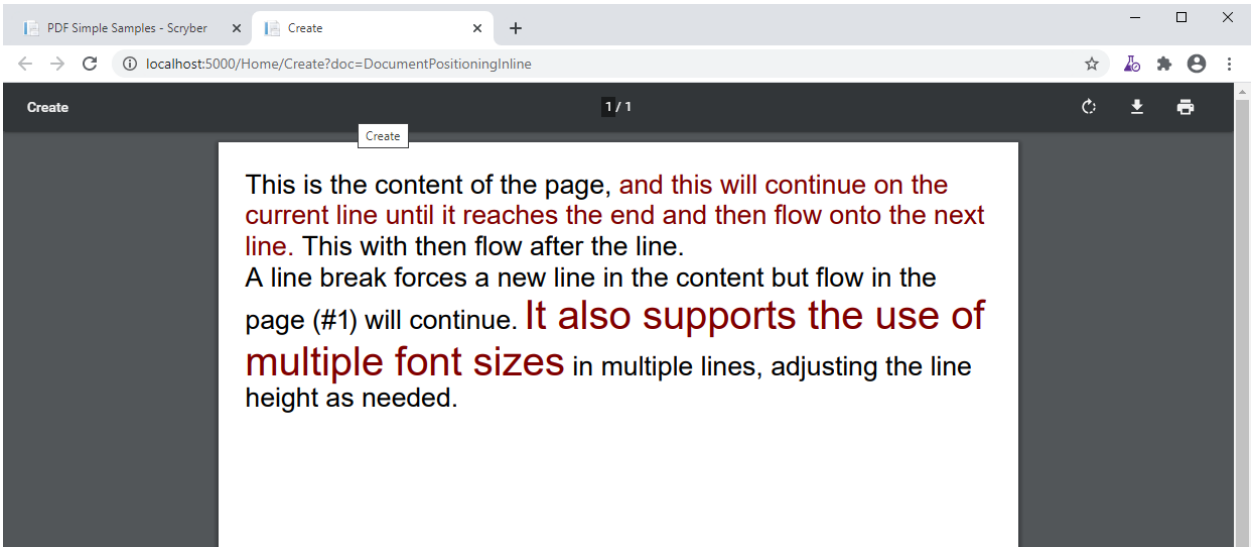
```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >

<Pages>

    <pdf:Page styles:margins="20pt" styles:font-size="20pt">
        <Content>
            This is the content of the page,
            <pdf:Span styles:fill-color="maroon" >and this will continue on the␣
↪current line until it reaches the end
            and then flow onto the next line.</pdf:Span>
```

(continues on next page)

(continued from previous page)

```
            This with then flow after the line.<pdf:Br/>
            A line break forces a new line in the content but flow in the page (#
↪<pdf:PageNumber />) will continue.
            <pdf:Span styles:fill-color="maroon" styles:font-size="30pt" >It also␣
↪supports the use of multiple font sizes</pdf:Span> in multiple lines,
            adjusting the line height as needed.
        </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```

Generating this document will create the following output (see *MVC Controller - Getting Started* or *Console or GUI - Getting Started* to understand how to do this).



For more information on laying out textual content see documenttextlayout

6.7.2 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.

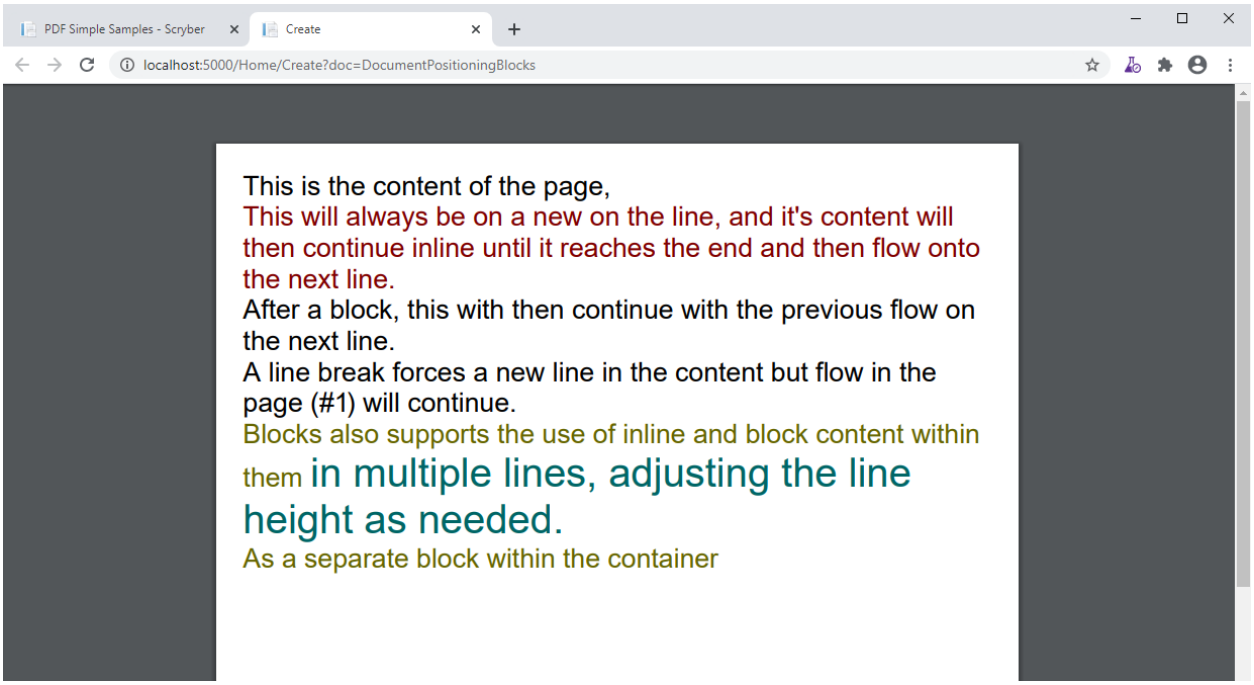
Examples of blocks are Div's, Paragraphs, Tables, BlockQuotes, Headings, Images, and Shapes.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >

    <Pages>

        <pdf:Page styles:margins="20pt" styles:font-size="20pt">
            <Content>
                This is the content of the page,
```

(continues on next page)

(continued from previous page)

```
                <pdf:Div styles:fill-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.</pdf:Div>

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

                <pdf:Div styles:fill-color="#666600" >
                Blocks also supports the use of inline and block content within them
                <pdf:Span styles:fill-color="#006666"  styles:font-size="30pt">in␣
↪multiple lines, adjusting the line height as needed.</pdf:Span>
                <pdf:Div >As a separate block within the container</pdf:Div>
                </pdf:Div>

            </Content>
        </pdf:Page>
    </Pages>

</pdf:Document>
```



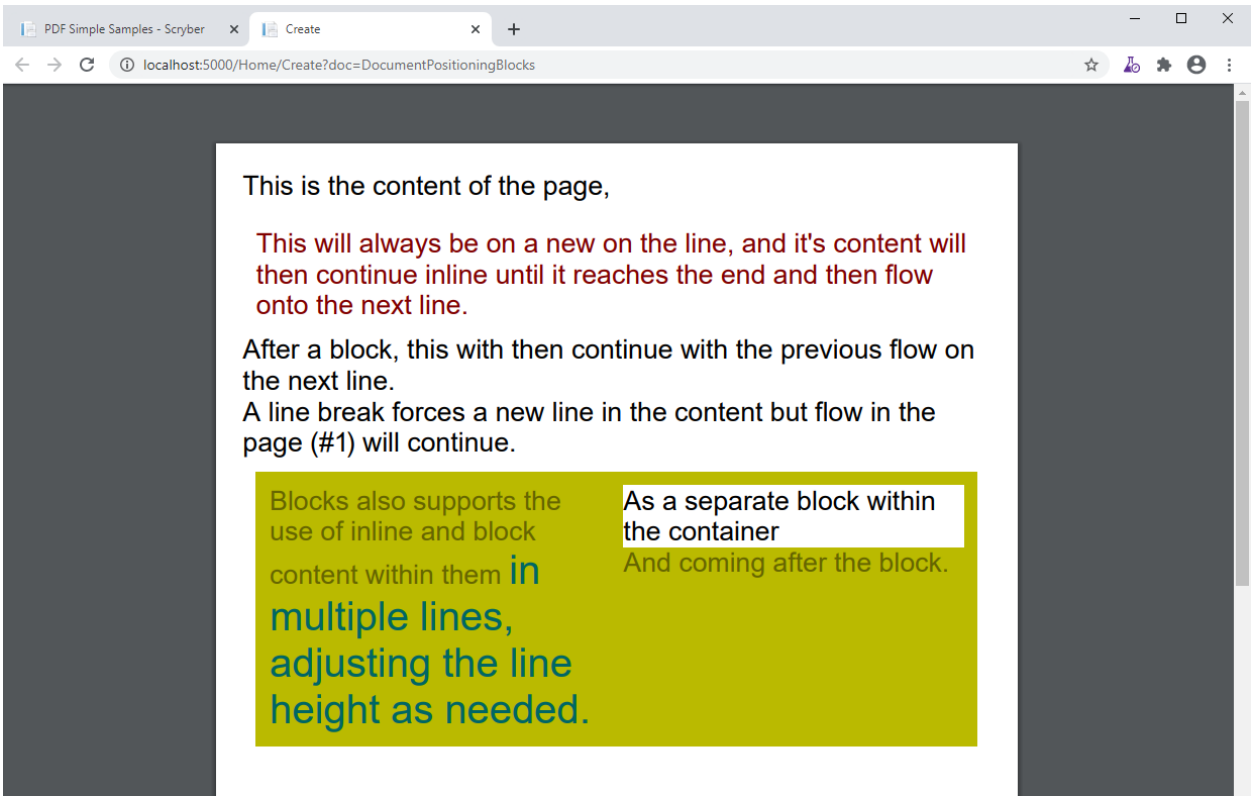
Blocks also support the use of backgrounds, borders, margins and padding. They also support *Flowing pages and columns*

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd" >

<Pages>
```

(continues on next page)

(continued from previous page)

```
    <pdf:Page styles:margins="20pt" styles:font-size="20pt">
    <Content>
        This is the content of the page,

        <pdf:Div styles:fill-color="maroon" styles:margins="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.
        </pdf:Div>

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

        <pdf:Div styles:fill-color="#666600" styles:bg-color="#BBBB00" styles:padding=
↪"10pt"
                 styles:margins="10pt" styles:column-count="2">
            Blocks also supports the use of inline and block content within them

            <pdf:Span styles:fill-color="#006666"  styles:font-size="30pt">in␣
↪multiple lines,
                adjusting the line height as needed.</pdf:Span>

            <!-- breaking onto a new column-->
            <pdf:ColumnBreak />

            <pdf:Div styles:fill-color="black" styles:bg-color="white" >As a separate␣
↪block within the container</pdf:Div>
            And coming after the child block.
        </pdf:Div>

    </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```



6.7.3 Changing the position-mode

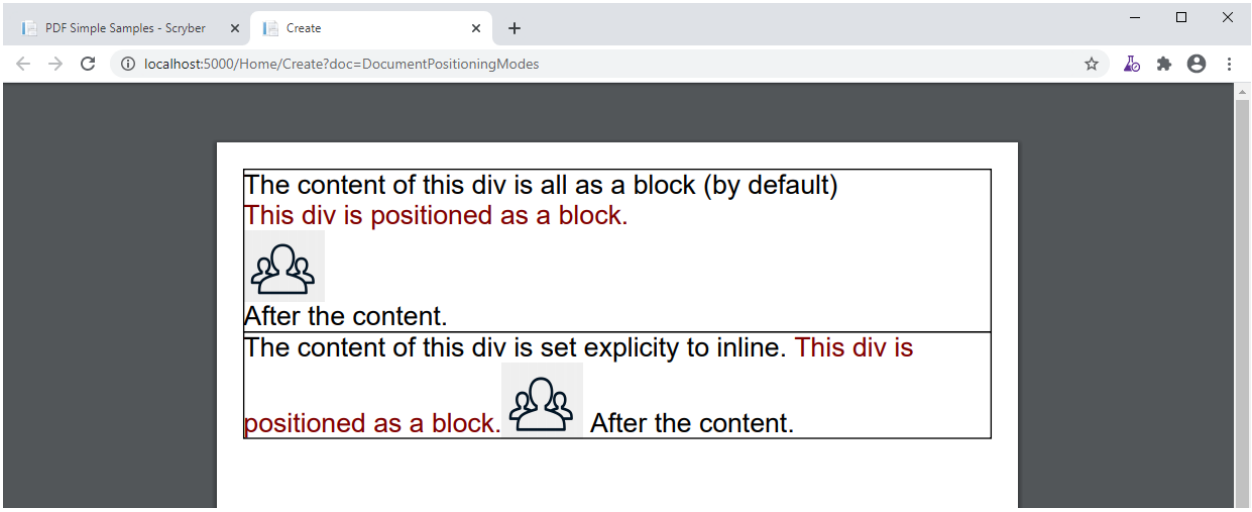
It is posible to change the default position mode for many components on the page. A span can be a block and a div can be a span. Images and shapes (see document_images and *Drawing paths and shapes - td*) also support the use of the the position mode.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd" >

<Pages>

    <pdf:Page styles:margins="20pt" styles:font-size="20pt">
        <Content>
            <pdf:Div styles:border-color="black" styles:border-width="1pt" >
                The content of this div is all as a block (by default)

                <pdf:Div styles:fill-color="maroon" >This div is positioned as a␣
↪block.</pdf:Div>

                <!-- Images are by detault displayed as blocks -->
                <pdf:Image styles:width="60pt" src="../../Content/Images/group.png" />

                After the content.
            </pdf:Div>
```

(continues on next page)

(continued from previous page)

```
            <pdf:Div styles:border-color="black" styles:border-width="1pt" >
                The content of this div is set explicity to inline.

                <pdf:Div styles:position-mode="Inline" styles:fill-color="maroon">
↪This div is positioned as a block.</pdf:Div>

                <!-- Image is also set to inline and will increase the line height␣
↪automatically -->
                <pdf:Image styles:position-mode="Inline" styles:width="60pt"  src="../
↪../Content/Images/group.png" />

                After the content.
            </pdf:Div>


        </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```



6.7.4 The full-width attribute

The attribute full-width makes any block component automatically fill the available width of the region. Even if the inner content does not need it. It's effectivly set as 100% width.

If it's set to false, the block will be as wide as needed (without going beyond the boundaries of it's own containing region). 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.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >

<Styles>
```

(continues on next page)

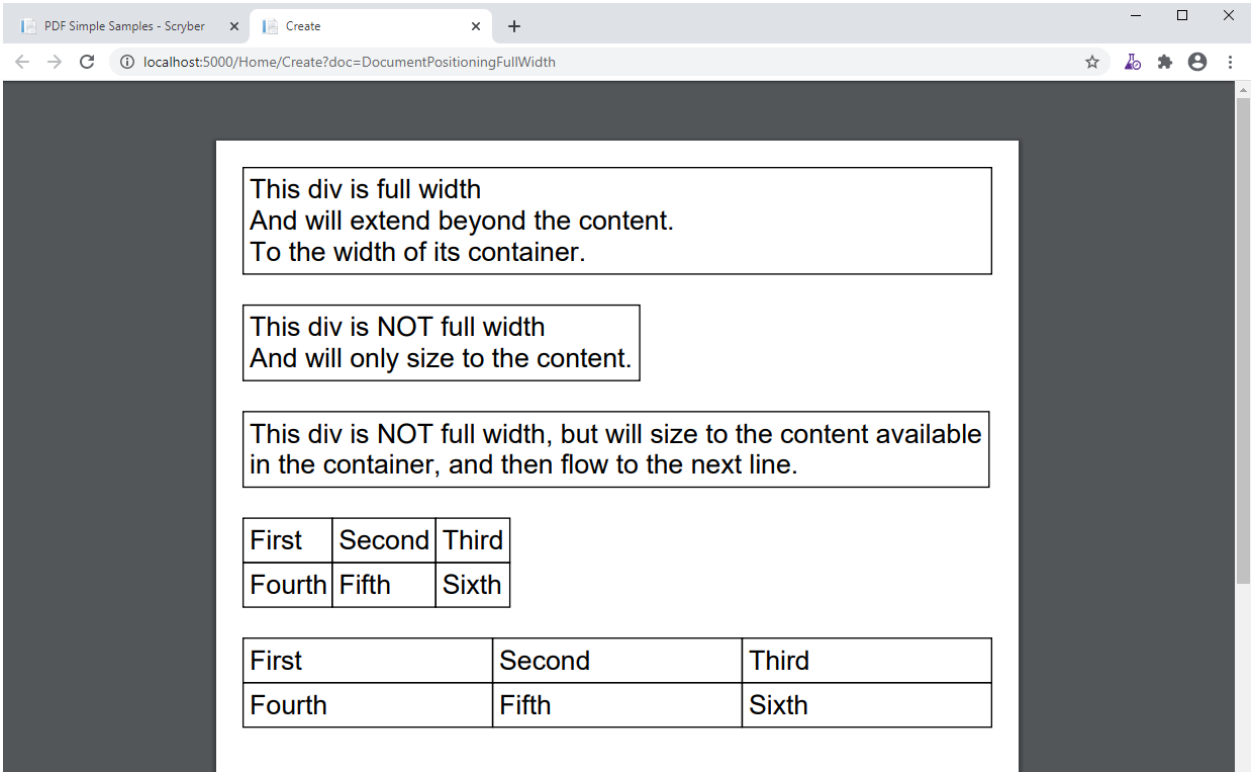
(continued from previous page)

```
    <!-- Using a style to repeat the border is easier -->
    <styles:Style applied-class="bordered" >
        <styles:Border color="black" style="Solid" width="1pt"/>
        <styles:Padding all="5pt"/>
    </styles:Style>
</Styles>
<Pages>

    <pdf:Page styles:margins="20pt" styles:font-size="20pt">
    <Content>
        <pdf:Div styles:class="bordered" >
            This div is full width<pdf:Br/>
            And will extend beyond the content.<pdf:Br/>
            To the width of its container.
        </pdf:Div>
        <pdf:Br/>
        <pdf:Div styles:class="bordered" styles:full-width="false" >
            This div is NOT full width<pdf:Br/>
            And will only size to the content.<pdf:Br/>
        </pdf:Div>
        <pdf:Br/>
        <pdf:Div styles:class="bordered" styles:full-width="false" >
            This div is NOT full width,
            but will  size to the content available in the container,
            and then flow to the next line.
        </pdf:Div>
        <pdf:Br/>
        <!-- Tables are not by default full width-->
        <pdf:Table>
            <pdf:Row>
                <pdf:Cell styles:class="bordered">First</pdf:Cell>
                <pdf:Cell styles:class="bordered">Second</pdf:Cell>
                <pdf:Cell styles:class="bordered">Third</pdf:Cell>
            </pdf:Row>
            <pdf:Row>
                <pdf:Cell styles:class="bordered">Fourth</pdf:Cell>
                <pdf:Cell styles:class="bordered">Fifth</pdf:Cell>
                <pdf:Cell styles:class="bordered">Sixth</pdf:Cell>
            </pdf:Row>
        </pdf:Table>
        <pdf:Br/>
        <!-- But can be set to full width explicitly or in styles -->
        <pdf:Table styles:full-width="true">
            <pdf:Row>
                <pdf:Cell styles:class="bordered">First</pdf:Cell>
                <pdf:Cell styles:class="bordered">Second</pdf:Cell>
                <pdf:Cell styles:class="bordered">Third</pdf:Cell>
            </pdf:Row>
            <pdf:Row>
                <pdf:Cell styles:class="bordered">Fourth</pdf:Cell>
                <pdf:Cell styles:class="bordered">Fifth</pdf:Cell>
                <pdf:Cell styles:class="bordered">Sixth</pdf:Cell>
            </pdf:Row>
        </pdf:Table>
    </Content>
    </pdf:Page>
</Pages>
```

(continues on next page)

(continued from previous page)

```
</pdf:Document>
```



For more on styles see *Styles in your template*

6.7.5 Flowing around components

At the moment scryber does not support flowing content around other components. It is something we are looking at supporting. If you want to help, please get in touch.

6.7.6 Relative Positioning

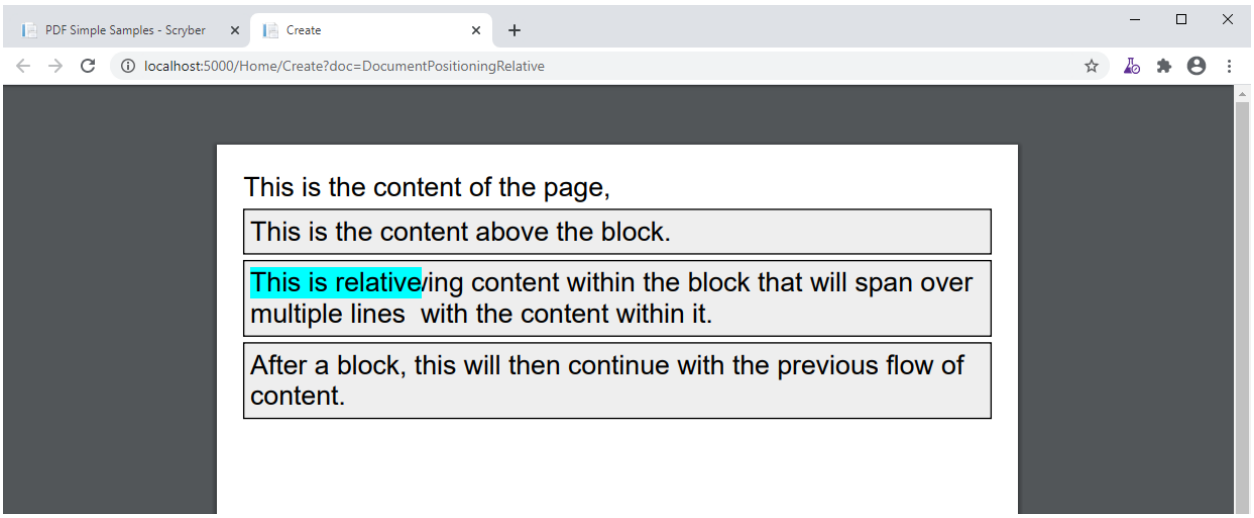
When you set the position-mode to Relative, it declares the position of that component relative to the block parent. The component will no longer be in the flow of any inline content, nor alter the layout of the following components.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >
<Styles>
    <styles:Style applied-class="bordered">
    <styles:Border color="black" style="Solid" width="1pt"/>
    <styles:Padding all="5pt"/>
    <styles:Background color="#AAAAAA" />
    <styles:Margins top="5pt"/>
    </styles:Style>
```

(continues on next page)

(continued from previous page)

```
</Styles>
<Pages>

    <pdf:Page styles:margins="20pt" styles:font-size="20pt">
    <Content>
        This is the content of the page,

        <pdf:Div styles:class="bordered" >This is the content above the block.</
↪pdf:Div>

        <pdf:Div styles:class="bordered" >This is the flowing content within the␣
↪block that will span over multiple lines
            <pdf:Span styles:position-mode="Relative" styles:bg-color="aqua" >This is␣
↪relative</pdf:Span>
            with the content within it.
        </pdf:Div>

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

    </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```



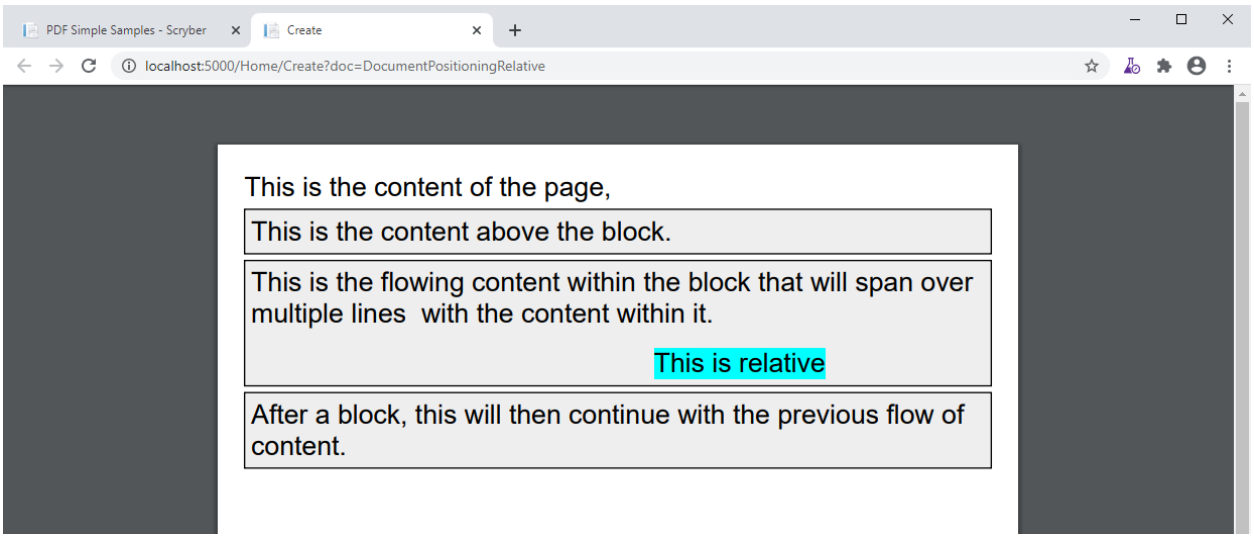
By default the position will be 0,0 (top, left), but using the x and y attributes it can be altered. The parent block will grow to accomodate the content including any of it's relatively positioned content. And push any content after the block down.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >
<Styles>
    <styles:Style applied-class="bordered">
```

(continues on next page)

(continued from previous page)

```
    <styles:Border color="black" style="Solid" width="1pt"/>
    <styles:Padding all="5pt"/>
    <styles:Background color="#AAAAAA"/>
    <styles:Margins top="5pt"/>
    </styles:Style>
</Styles>
<Pages>

    <pdf:Page styles:margins="20pt" styles:font-size="20pt">
    <Content>
        This is the content of the page,

        <pdf:Div styles:class="bordered" >This is the content above the block.</
↪pdf:Div>

        <pdf:Div styles:class="bordered" >This is the flowing content within the␣
↪block that will span over multiple lines
            <pdf:Span styles:position-mode="Relative" styles:bg-color="aqua" styles:x=
↪"300pt" styles:y="60pt" >This is relative</pdf:Span>
            with the content within it.
        </pdf:Div>

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

    </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```

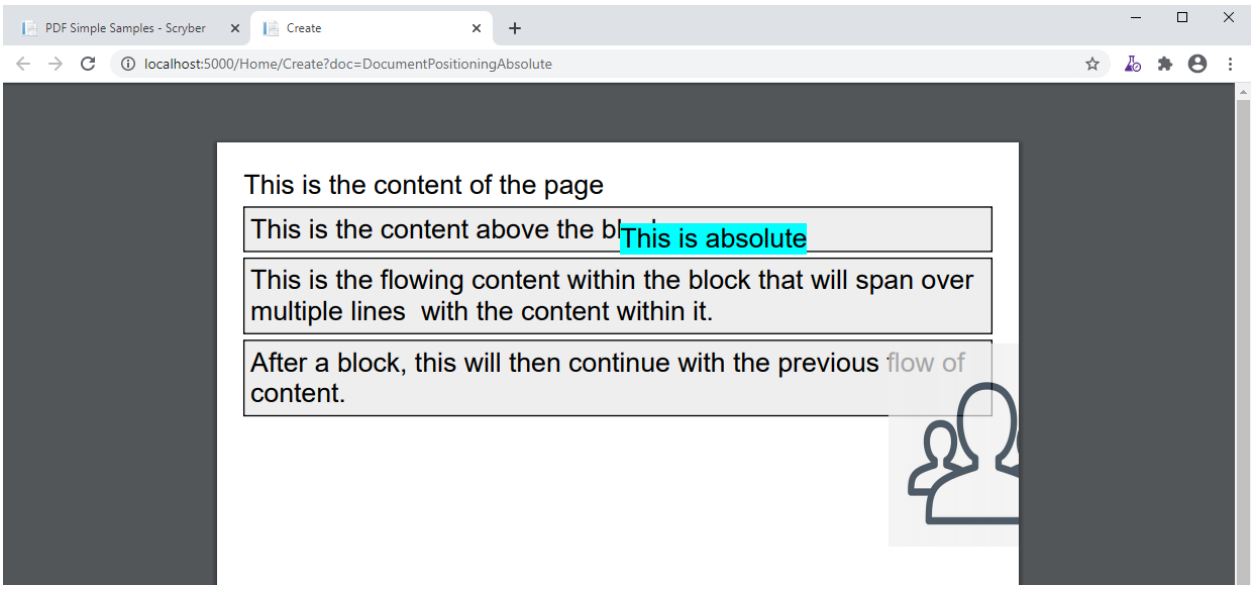


6.7.7 Absolute Positioning

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" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
               xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >
<Styles>
    <styles:Style applied-class="bordered">
    <styles:Border color="black" style="Solid" width="1pt"/>
    <styles:Padding all="5pt"/>
    <styles:Background color="#AAAAAA" />
    <styles:Margins top="5pt"/>
    </styles:Style>
</Styles>
<Pages>

    <pdf:Page styles:margins="20pt" styles:font-size="20pt">
    <Content>
        This is the content of the page

        <pdf:Div styles:class="bordered" >This is the content above the block.</
↪pdf:Div>

        <pdf:Div styles:class="bordered" >This is the flowing content within the␣
↪block that will span over multiple lines
            <!-- Absolutely positioned content -->
            <pdf:Span styles:position-mode="Absolute" styles:bg-color="aqua" styles:x=
↪"300pt" styles:y="60pt" >This is absolute</pdf:Span>
            with the content within it.
        </pdf:Div>

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

        <!-- Absolute postitioning can be applied to any component, and size can be␣
↪specified. -->
        <pdf:Image styles:position-mode="Absolute" src="../../Content/Images/group.png
↪" styles:fill-opacity="0.7"
               styles:x="500pt" styles:y="150pt" styles:width="150pt" styles:height=
↪"150pt" />
    </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```



6.7.8 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*

If no units are specified then the default is points. See *Document drawing units and measures - td* for more information.

6.7.9 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.

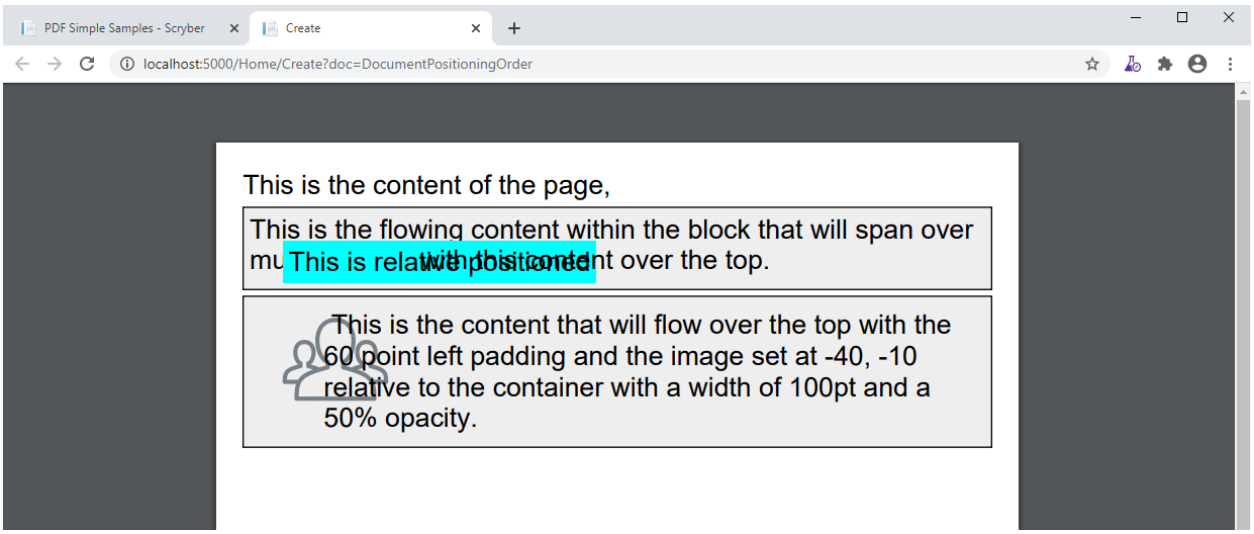
```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >
<Styles>
    <styles:Style applied-class="bordered">
    <styles:Border color="black" style="Solid" width="1pt"/>
    <styles:Padding all="5pt"/>
    <styles:Background color="#AAAAAA" opacity="0.2"/>
    <styles:Margins top="5pt"/>
    </styles:Style>
</Styles>
<Pages>
```

(continues on next page)

(continued from previous page)

```
    <pdf:Page styles:margins="20pt" styles:font-size="20pt">
    <Content>
        This is the content of the page,

        <pdf:Div styles:class="bordered" >This is the flowing content within the␣
↪block that
            will span over multiple lines
            <pdf:Span styles:position-mode="Relative" styles:bg-color="aqua" styles:x=
↪"25pt"
                        styles:y="20pt" styles:padding="4pt" >This is relative␣
↪positioned</pdf:Span>
            with this content over the top.
        </pdf:Div>

        <pdf:Div styles:class="bordered" styles:padding="10 60 10 10">
            <pdf:Image src="../../Content/Images/group.png" styles:position-mode=
↪"Relative"
                        styles:x="-40pt" styles:y="-10pt" styles:width="100pt"␣
↪styles:fill-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.
        </pdf:Div>

    </Content>
    </pdf:Page>
</pages>

</pdf:Document>
```

By using this rule interesting effects can be designed.



6.7.10 Position z-index

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

6.7.11 Positioned components

There are 2 components that take advantage of the positioning within Scryber.

1. reference/pdf_canvas positions all direct child components in the canvas as relative, whether they have been decared as such or not.
2. reference/pdf_layergroup has a collection of child Layers. These will be relatively positioned to the group.

6.8 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.

6.8.1 Width and Height

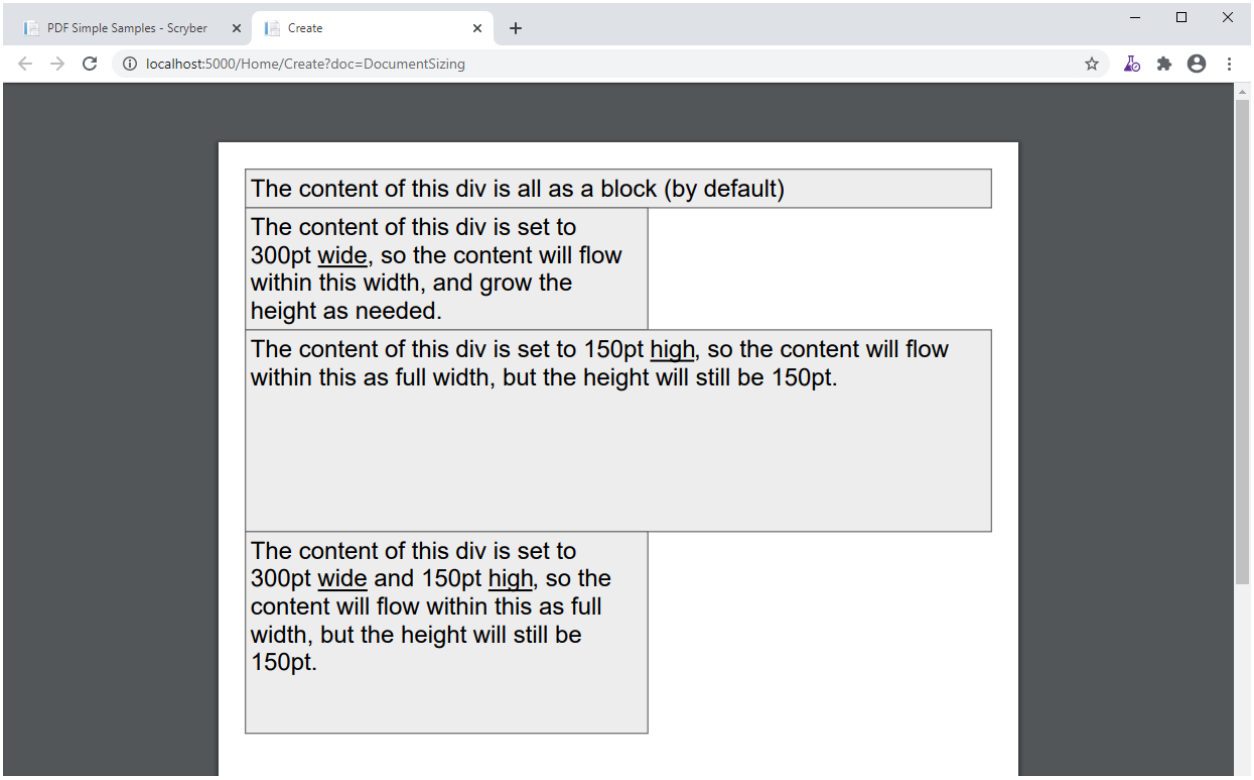
All block components support an explicit width and / or height value. If it's width is set, then any full-width style will be ignored.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >

<Styles>
    <styles:Style applied-class="bordered" >
        <styles:Border color="#777" width="1pt" style="Solid"/>
        <styles:Background color="#EEE"/>
        <styles:Padding all="4pt"/>
    </styles:Style>
</Styles>
<Pages>

    <pdf:Page styles:margins="20pt" styles:font-size="18pt">
        <Content>
            <pdf:Div styles:class="bordered" >
            The content of this div is all as a block (by default)
            </pdf:Div>

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

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

(continues on next page)

(continued from previous page)

```
            <pdf:Div styles:class="bordered" styles:width="300pt" styles:height="150pt
↪" >
            The content of this div is set to 300pt <pdf:U>wide</pdf:U> and 150pt
↪<pdf:U>high</pdf:U>, so the content will flow within this
            as full width, but the height will still be 150pt.
            </pdf:Div>

        </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```



6.8.2 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.

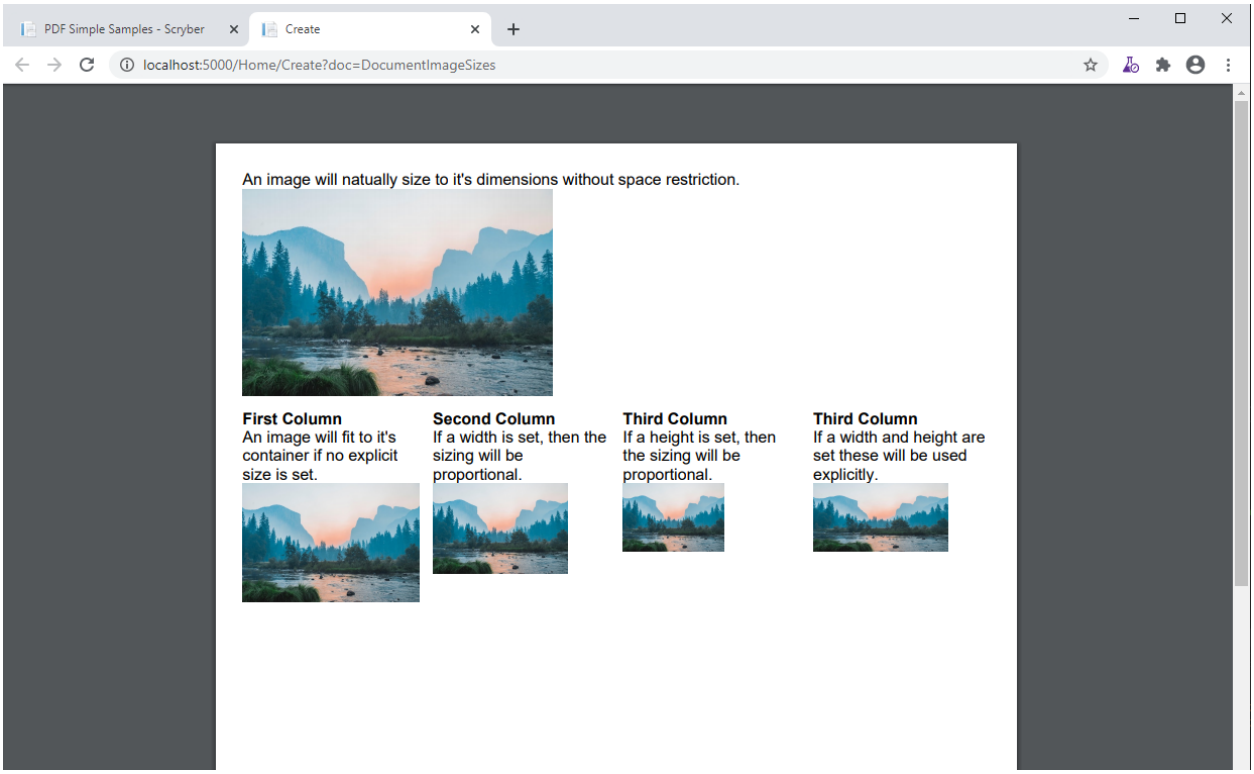
If both a width **and** height are assigned, then they will both be used to fit the image to that space. No matter what the originals' proportions are.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
```

(continues on next page)

(continued from previous page)

```
                xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >
<Pages>
    <pdf:Page styles:margins="20pt" styles:font-size="12pt" >
        <Content>

            <pdf:Span >An image will natually size to it's dimensions without space␣
↪restriction.</pdf:Span>
            <pdf:Image src="../../Content/Images/landscape.jpg" />

            <pdf:Div styles:column-count="4" styles:margins="10 0 0 0" >
                <pdf:B>First Column</pdf:B><pdf:Br/>
                An image will fit to it's container if no explicit size is set.
                <pdf:Image src="../../Content/Images/landscape.jpg" />
                <pdf:ColumnBreak/>
                <pdf:B>Second Column</pdf:B><pdf:Br/>
                If a width is set, then the sizing will be proportional.
                <pdf:Image src="../../Content/Images/landscape.jpg" styles:width=
↪"100pt" />
                <pdf:ColumnBreak/>
                <pdf:B>Third Column</pdf:B><pdf:Br/>
                If a height is set, then the sizing will be proportional.
                <pdf:Image src="../../Content/Images/landscape.jpg" styles:height=
↪"50pt" />

                <pdf:ColumnBreak/>
                <pdf:B>Third Column</pdf:B><pdf:Br/>
                If a width and height are set these will be used explicitly.
                <pdf:Image src="../../Content/Images/landscape.jpg" styles:width=
↪"100pt" styles:height="50pt" />
            </pdf:Div>

            <!-- Photo by Bailey Zindel on Unsplash -->
        </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```



6.8.3 Page Sizes

Pages are generally sized differently to components on a page, as they use the standard ISO and Imperial page enumeration. But they can also be a custom size. See *Pages and Sections* for details on how to alter the size of pages.

6.8.4 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 *Margins* and *Padding* style have 5 properties that can be set.

- All
- 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 elements can 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.

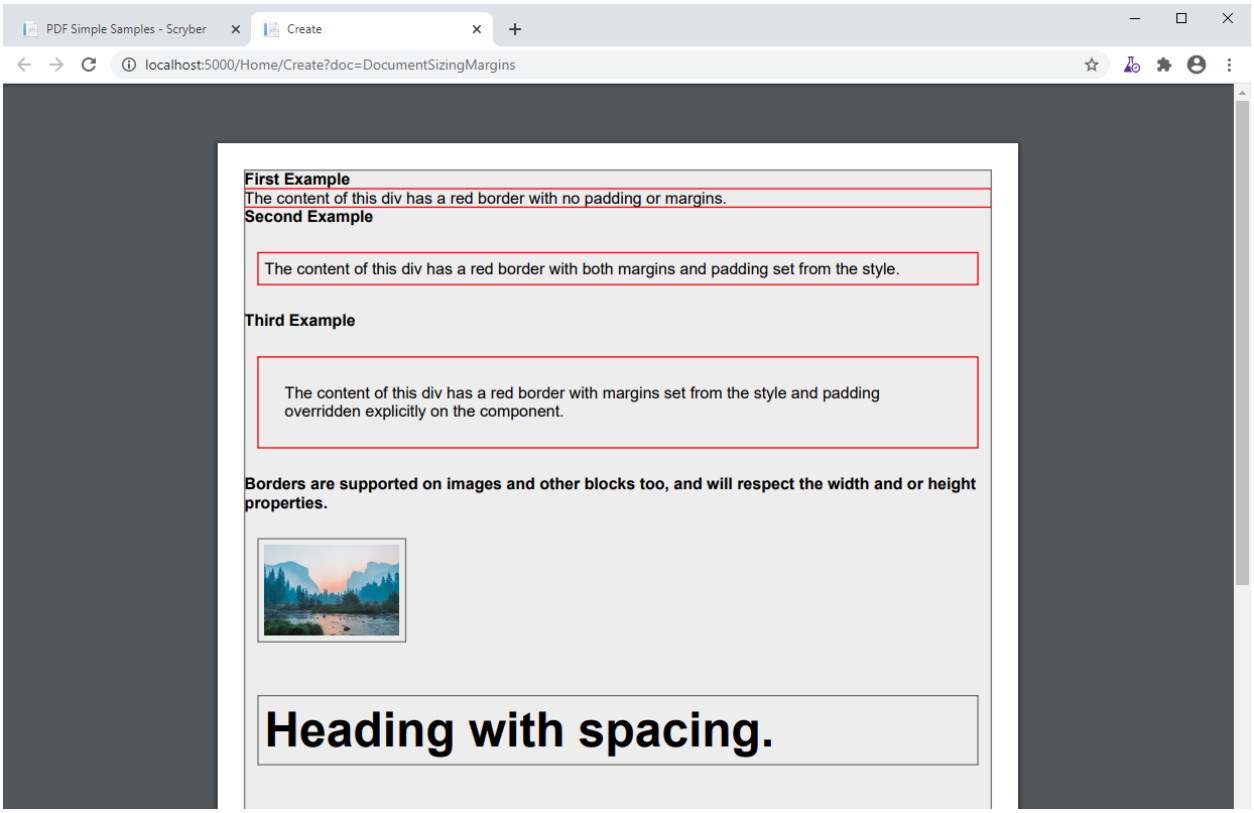
If not set then the values will be zero.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >

<Styles>

    <styles:Style applied-type="pdf:Page" >
        <styles:Font size="12pt"/>
        <styles:Margins all="20pt"/>
    </styles:Style>

    <styles:Style applied-class="bordered" >
        <styles:Border color="#777" width="1pt" style="Solid"/>
        <styles:Background color="#EEE"/>
    </styles:Style>

    <styles:Style applied-class="red">
        <styles:Border color="red"/>
    </styles:Style>

    <styles:Style applied-class="spaced" >
        <styles:Margins all="20pt" left="10pt" right="10pt"/>
        <styles:Padding all="5pt"/>
    </styles:Style>

</Styles>
<Pages>

    <pdf:Page styles:class="bordered" > <!--Styles applied to the page type -->
        <Content>
            <pdf:B>First Example</pdf:B>
            <pdf:Div styles:class="bordered red" >
                The content of this div has a red border with no padding or margins.
            </pdf:Div>

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

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

            <pdf:B>Borders are supported on images and other blocks too, and will␣
↪respect the width and or height properties.</pdf:B>
            <pdf:Image src="../../Content/Images/landscape.jpg" styles:class=
↪"bordered spaced" styles:width="100pt" />
```

(continues on next page)

(continued from previous page)

```
            <pdf:H1 styles:class="bordered spaced">Heading with spacing.</pdf:H1>
        </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```



6.8.5 Clipping

The block level components also support the use of a clipping (with overflow action) to reduce the size of the visible area within the block By default, content is truncated when an explicit size is reached. It cannot overflow, because of the size, so is truncated. When the overflow action is set to Clip, however, all the inner content of the block will be rendered, but effectively in a window on top of the content. The content outside the view of the window is still there, but not visible.

Along with the overflow action on a style a clipping can be applied in the same way as margins and padding. This will alter the 'size of the window' that content is seen through.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >

<Styles>

    <styles:Style applied-type="pdf:Page" >
```

(continues on next page)

(continued from previous page)

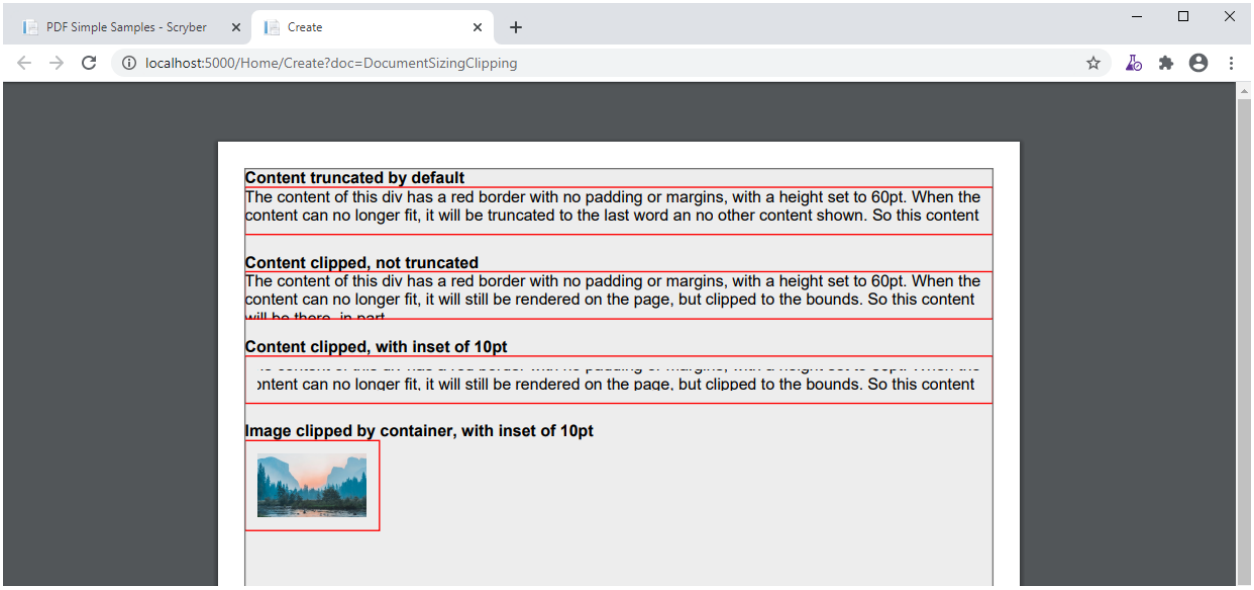
```
        <styles:Font size="12pt"/>
        <styles:Margins all="20pt"/>
    </styles:Style>

    <styles:Style applied-class="bordered" >
        <styles:Border color="#777" width="1pt" style="Solid"/>
        <styles:Background color="#EEE"/>
    </styles:Style>

    <styles:Style applied-class="red">
        <styles:Border color="red"/>
    </styles:Style>

    <!-- Our clipping style applies 10pt all around.
         It's NOT the same as padding.  -->

    <styles:Style applied-class="clipped" >
        <styles:Clipping all="10pt"/>
        <styles:Overflow action="Clip"/>
    </styles:Style>

</Styles>
<Pages>

    <pdf:Page styles:class="bordered" > <!--Styles applied to the page type -->
        <Content>
            <pdf:B>Content truncated by default</pdf:B>
            <pdf:Div styles:class="bordered red" styles:height="35pt" >
                The content of this div has a red border with no padding or margins,␣
↪with a height set to 60pt. When the content can no longer fit,
                it will be truncated to the last word an no other content shown. So␣
↪this content will not be visible, as it cannot be completely laid out.
            </pdf:Div>
            <pdf:Br/>
            <pdf:B>Content clipped, not truncated</pdf:B>
            <pdf:Div styles:class="bordered red" styles:height="35pt" styles:overflow-
↪action="Clip" >
                The content of this div has a red border with no padding or margins,␣
↪with a height set to 60pt. When the content can no longer fit,
                it will still be rendered on the page, but clipped to the bounds. So␣
↪this content will be there, in part.
            </pdf:Div>

            <pdf:Br/>
            <pdf:B>Content clipped, with inset of 10pt</pdf:B>
            <pdf:Div styles:class="bordered red clipped" styles:height="35pt" >
                The content of this div has a red border with no padding or margins,␣
↪with a height set to 60pt. When the content can no longer fit,
                it will still be rendered on the page, but clipped to the bounds. So␣
↪this content will be there, in part.
            </pdf:Div>

            <pdf:Br/>
            <pdf:B>Image clipped by container, with inset of 10pt</pdf:B>
            <pdf:Div styles:class="bordered red clipped" styles:width="100pt" >
                <pdf:Image src="../../Content/Images/landscape.jpg" />
            </pdf:Div>
```

(continues on next page)

(continued from previous page)

```
        </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```

Note: The clipping only applies to the inner content. It's effectively drawn and then clipped to shape. This means that clipping directly on images is not supported.



6.8.6 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.

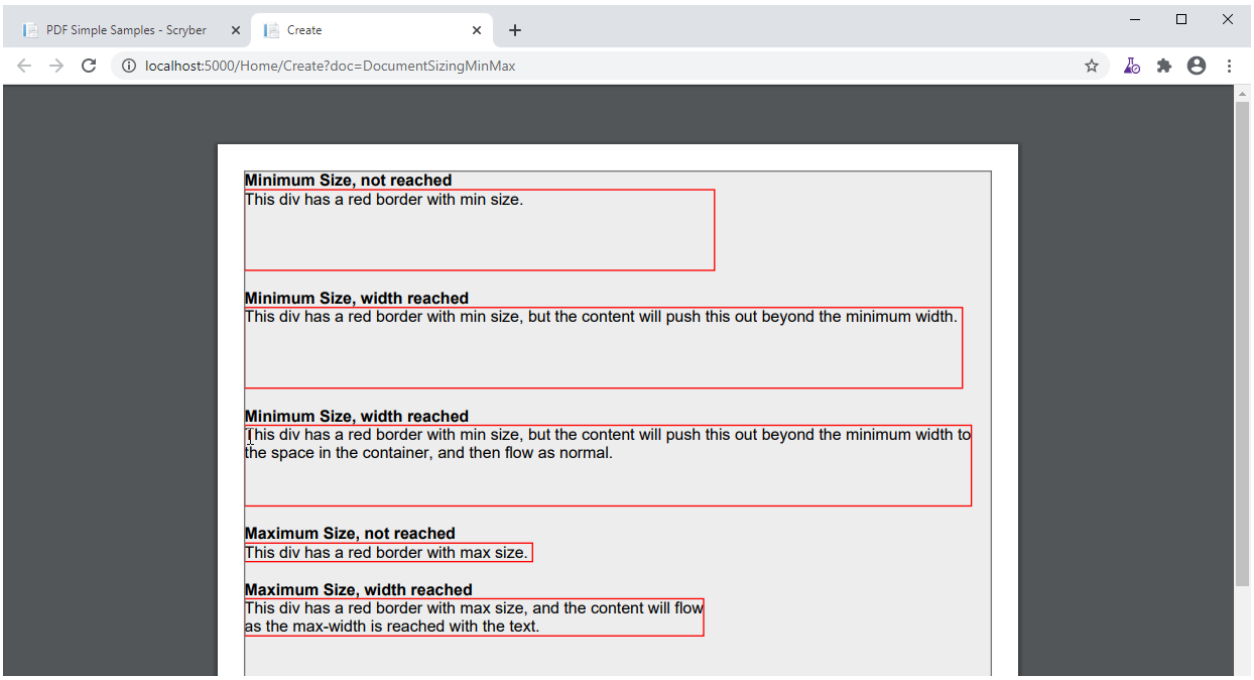
As you might expect, the minimum will ensure that a container is at least as big as the specified value, and that the maximum will ensure the content, never grows beyond that specified value.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd" >

<Styles>

    <styles:Style applied-type="pdf:Page" >
        <styles:Font size="12pt"/>
        <styles:Margins all="20pt"/>
    </styles:Style>

    <styles:Style applied-class="bordered" >
        <styles:Border color="#777" width="1pt" style="Solid"/>
        <styles:Background color="#EEE"/>
```

(continues on next page)

(continued from previous page)

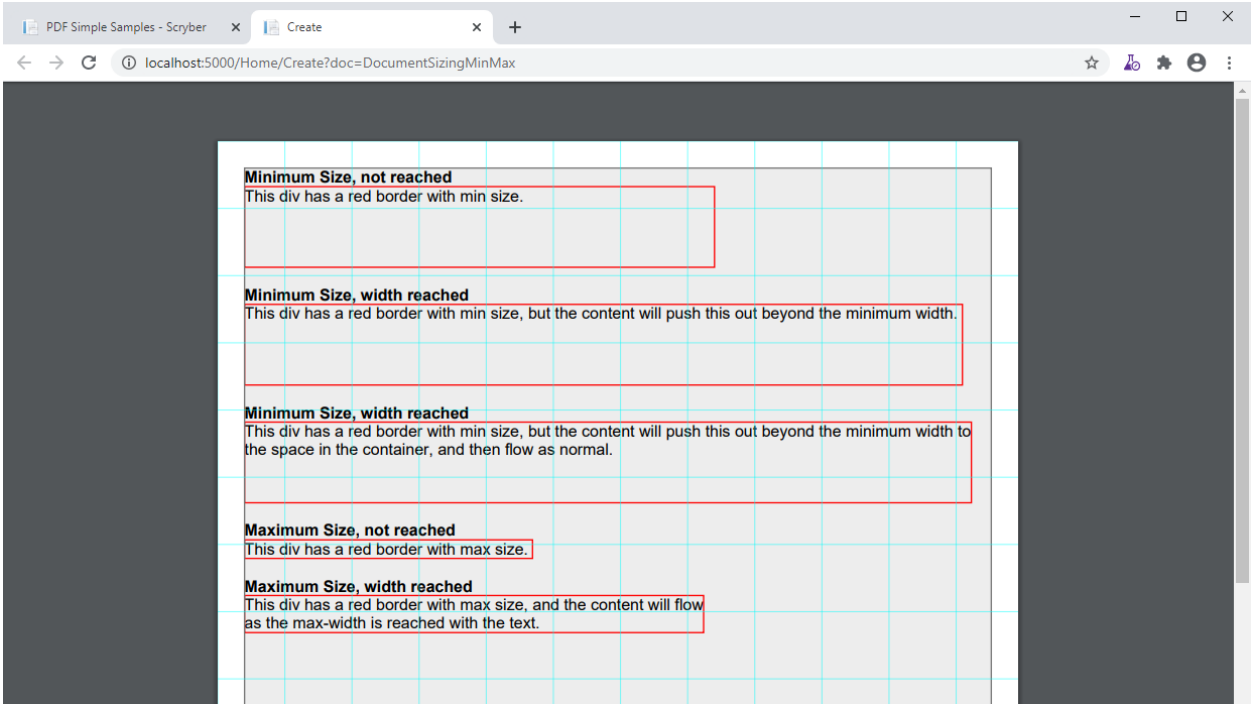
```
    </styles:Style>

    <styles:Style applied-class="red">
        <styles:Border color="red"/>
    </styles:Style>

    <styles:Style applied-class="sized" >
        <styles:Size full-width="false" max-height="60pt" max-width="350pt"/>
    </styles:Style>

</Styles>
<Pages>

    <pdf:Page styles:class="bordered" > <!--Styles applied to the page type -->
    <Content>
        <pdf:B>Minimum Size, not reached</pdf:B>
        <pdf:Div styles:class="bordered red" styles:full-width="false" styles:min-
↪height="60pt" styles:min-width="350pt" >
            This div has a red border with min size.
        </pdf:Div>

        <pdf:Br/>
        <pdf:B>Minimum Size, width reached</pdf:B>
        <pdf:Div styles:class="bordered red" styles:full-width="false" styles:min-
↪height="60pt" styles:min-width="350pt" >
            This div has a red border with min size, but the content will push this␣
↪out beyond the minimum width.
        </pdf:Div>

        <pdf:Br/>
        <pdf:B>Minimum Size, width reached</pdf:B>
        <pdf:Div styles:class="bordered red" styles:full-width="false" styles:min-
↪height="60pt" styles: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.
        </pdf:Div>

        <pdf:Br/>
        <pdf:B>Maximum Size, not reached</pdf:B>
        <pdf:Div styles:class="bordered red sized" >
            This div has a red border with max size.
        </pdf:Div>

        <pdf:Br/>
        <pdf:B>Maximum Size, width reached</pdf:B>
        <pdf:Div styles: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.
        </pdf:Div>

    </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```



6.8.7 Sizing Grid

In order to visually measure your sizes, position and content - Scryber supports the use of an overlay grid. This can only be set on a style, rather than inline to components. But it does have the ability for position, spacing and offsets.

```
<styles:Style applied-type="pdf:Page" >
  <styles:Font size="12pt"/>
  <styles:Margins all="20pt"/>
  <styles:Overlay-Grid color="aqua" spacing="50pt" show="true"/>
</styles:Style>
```



6.9 Pages and Sections

All the visual content in a document sit's in pages. Scryber supports the use of both a single *Scryber.Components.PDFPage* with content within it. And mutliple flowing pages in a reference/pdf_section.

6.9.1 A Page and its content

A single page has a structure of optional elements

- Header - Optional, but always sited at the top of a page
- Content - Sited between the Header and Footer.
- Footer - Optional, but always sited at the bottom of a page

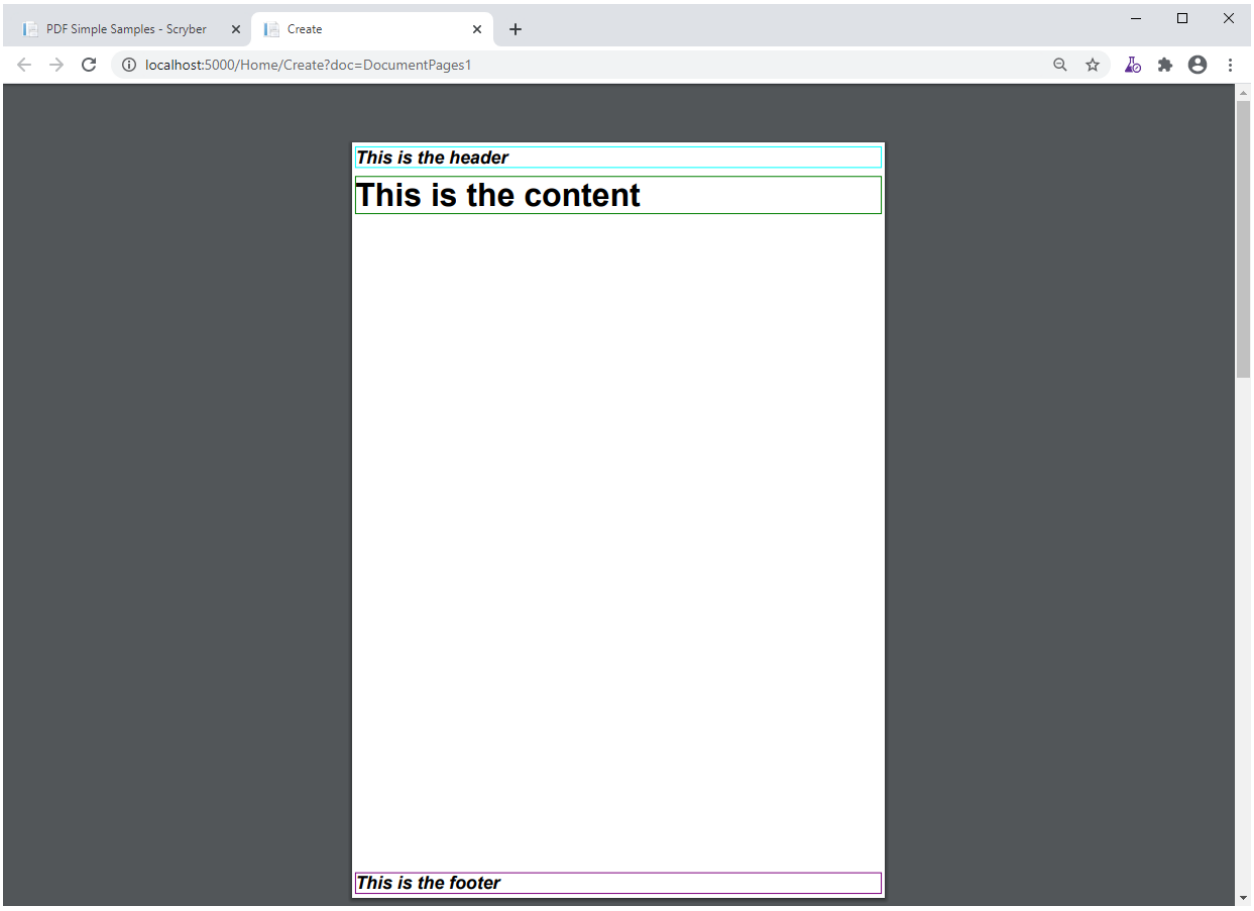
If a page has a header or footer the available space for the content will be reduced.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
              xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd" >
<Pages>
    <pdf:Page >
        <Header>
            <pdf:H4 styles:margins="5pt" styles:border-width="1pt" styles:border-
↪color="aqua" >This is the header</pdf:H4>
        </Header>
        <Content>
```

(continues on next page)

(continued from previous page)

```
            <pdf:H1 styles:margins="5pt" styles:border-width="1pt" styles:border-
↪color="green" >This is the content</pdf:H1>
        </Content>
        <Footer>
            <pdf:H4 styles:margins="5pt" styles:border-width="1pt" styles:border-
↪color="purple" >This is the footer</pdf:H4>
        </Footer>
    </pdf:Page>
</Pages>

</pdf:Document>
```



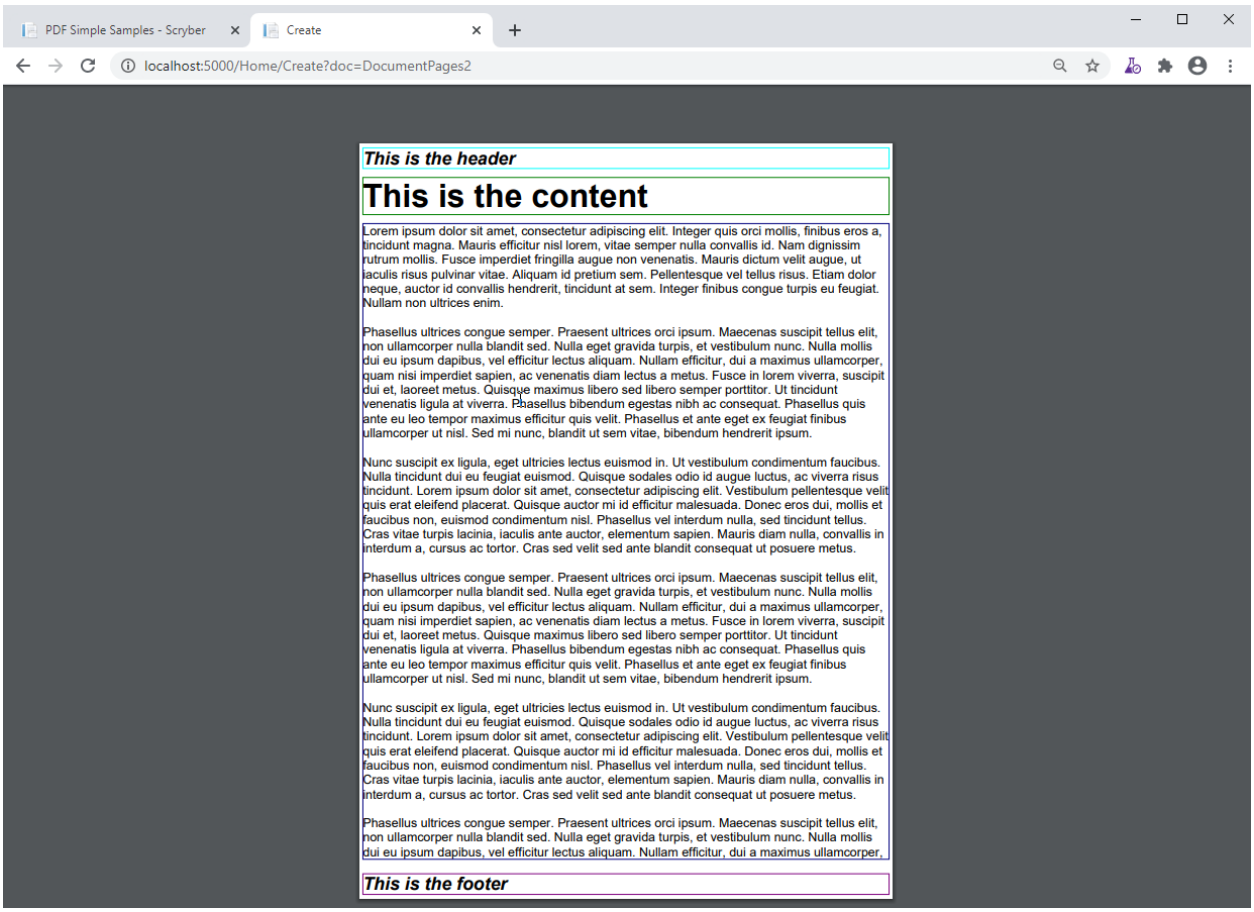
If the size of the content is more than can fit on a page it will be truncated.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
            xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
            xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd" >
<Pages>
    <pdf:Page>
        <Header>
            <pdf:H4 styles:margins="5pt" styles:border-width="1pt" styles:border-
↪color="aqua" >This is the header</pdf:H4>
```

(continues on next page)

(continued from previous page)

```
        </Header>
        <Content>
            <pdf:H1 styles:margins="5pt" styles:border-width="1pt" styles:border-
↪color="green" >This is the content</pdf:H1>
            <pdf:Div styles:margins="5pt" styles:font-size="14pt" styles:border-width=
↪"1pt" styles: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.<pdf:Br/>
            <pdf:Br/>
            <!-- Truncated for brevity
            .
            . -->
            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.<pdf:Br/>
        </pdf:Div>
        </Content>
        <Footer>
            <pdf:H4 styles:margins="5pt" styles:border-width="1pt" styles:border-
↪color="purple" >This is the footer</pdf:H4>
        </Footer>
    </pdf:Page>
</Pages>

</pdf:Document>
```



6.9.2 Sections and continuation

A section differs from a page in 2 ways. Firstly the default style has an overflow action of NewPage (rather than Truncate), and it also has allows for a definition of a continuation header and footer.

If defined, then the continuation headers and footers will be shown on the following pages, after the first. If not defined, then the main page headers and footers will be shown.

So if we change our *pdf:Page* element to a *pdf:Section* we can add a continuation header and flow onto multiple pages.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
              xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd" >
<Pages>
    <pdf:Section>
        <Header>
            <pdf:H4 styles:margins="5pt" styles:border-width="1pt" styles:border-
↪color="aqua" >This is the header</pdf:H4>
        </Header>
        <Continuation-Header>
            <pdf:H4 styles:margins="5pt" styles:border-width="1pt" styles:border-
↪color="fuschia" >This is the continuation header</pdf:H4>
```

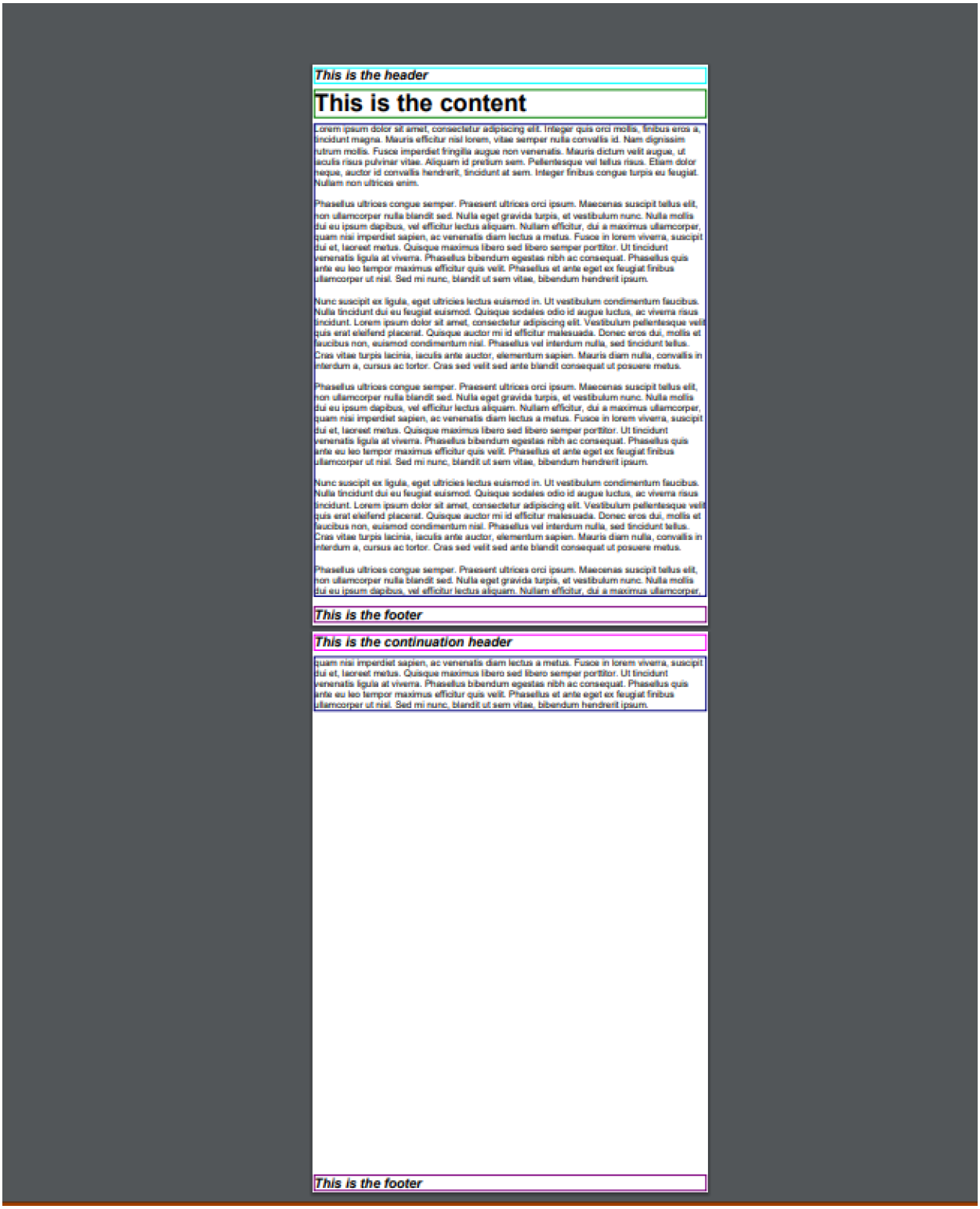
(continues on next page)

(continued from previous page)

```
        </Continuation-Header>
        <Content>
            <pdf:H1 styles:margins="5pt" styles:border-width="1pt" styles:border-
↪color="green" >This is the content</pdf:H1>
            <pdf:Div styles:margins="5pt" styles:font-size="14pt" styles:border-width=
↪"1pt" styles: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.<pdf:Br/>
            <pdf:Br/>
            <!-- Truncated for brevity
            .
            . -->
            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.<pdf:Br/>
        </pdf:Div>
        </Content>
        <Footer>
            <pdf:H4 styles:margins="5pt" styles:border-width="1pt" styles:border-
↪color="purple" >This is the footer</pdf:H4>
        </Footer>
    </pdf:Section>
</Pages>

</pdf:Document>
```

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

The footer is consistent throughout, so shows on both output pages.



6.9.3 Page breaks

When using an overflowing section it's possible to explicitly force a break in the pages using the *pdf:PageBreak* component. This can appear within any block, and will force all the other parent components to stop their layout on

the current layout page, and move to the next layout page. Borders, margins and padding will (should) be preserved.

As with other components, it is also possible to bind the visibility of a page break too. If it's visible then the break will occur if not then thecontent will flow as normal.

see *Page size and orientation* below for an example of using a page break.

6.9.4 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 page or via styles to the standard ISO or Imperial page sizes, in landscape or portrait, or even a custom size.

- **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,

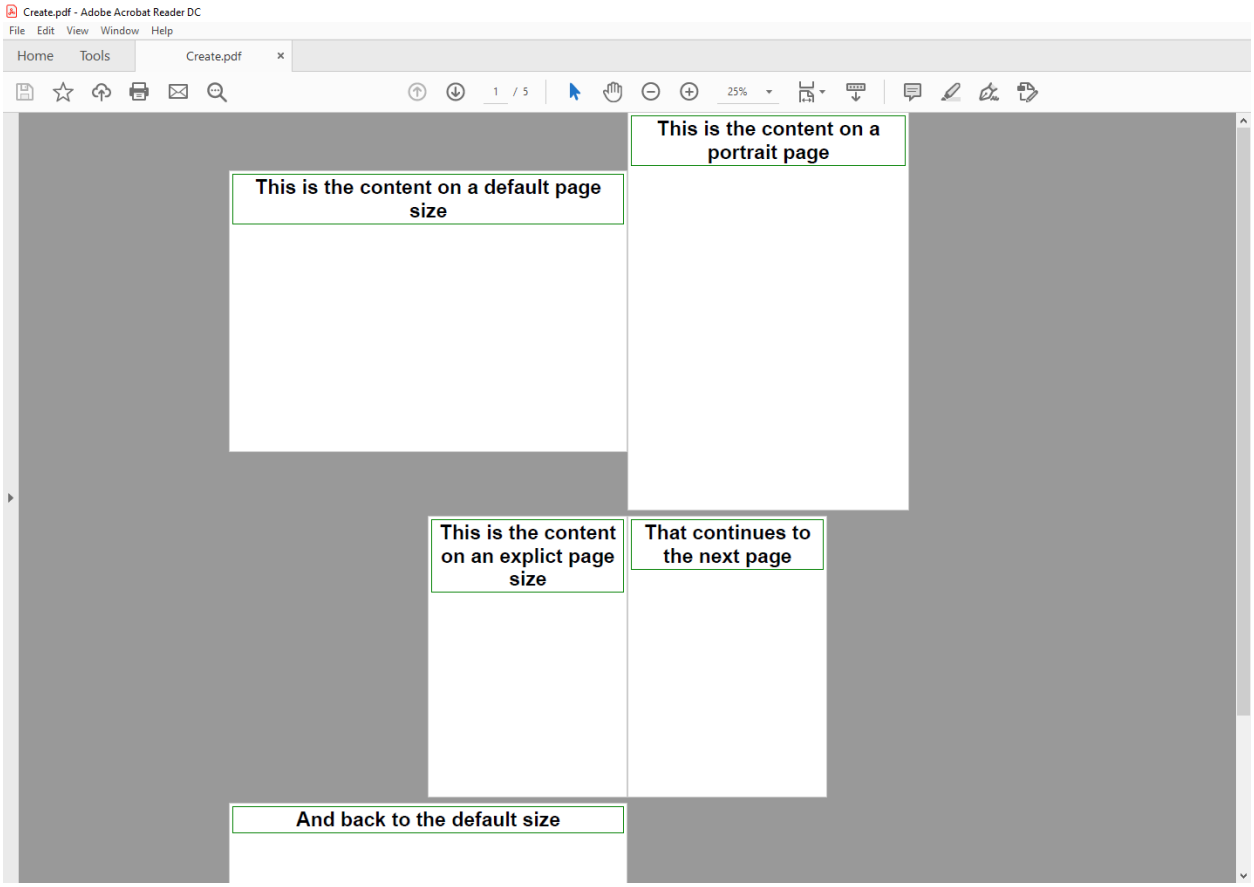
A section can only be 1 size of paper, but different sections and different pages can have different sizes.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
             xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
             xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd" >
<Styles>

    <!-- changing the default page size to A3 Landscape -->
    <styles:Style applied-type="pdf:Page" >
    <styles:Page size="A3" orientation="Landscape"/>
    </styles:Style>

    <!-- a style for portrait pages-->
    <styles:Style applied-class="long" >
    <styles:Page orientation="Portrait"/>
    </styles:Style>

    <!-- set up the default style for a heading 1-->
    <styles:Style applied-type="pdf:H1" >
    <styles:Border color="green" width="2"/>
    <styles:Padding all="5pt"/>
    <styles:Margins all="10pt"/>
    <styles:Font size="60pt"/>
    <styles:Position h-align="Center"/>
    </styles:Style>
</Styles>

<Pages>
    <pdf:Page>
    <Content>
```

(continues on next page)

(continued from previous page)

```
        <pdf:H1>This is the content on a default page size</pdf:H1>
    </Content>
    </pdf:Page>

    <pdf:Page styles:class="long">
    <Content>
        <pdf:H1>This is the content on a portrait page</pdf:H1>
    </Content>
    </pdf:Page>

    <pdf:Section styles:class="long" styles:paper-size="A4">
    <Content>
        <pdf:H1>This is the content on an explict page size</pdf:H1>
        <!-- Force a break in the page -->
        <pdf:PageBreak/>
        <pdf:H1 >That continues to the next page</pdf:H1>
    </Content>
    </pdf:Section>

    <pdf:Section>
    <Content>
        <pdf:H1>And back to the default size</pdf:H1>
    </Content>
    </pdf:Section>
</Pages>

</pdf:Document>
```



6.9.5 Page Groups

The *pdf:PageGroup* allows for consistency across a set of pages. They will pass styles, page numbers, parameters, headers etc. down to any pages within the group.

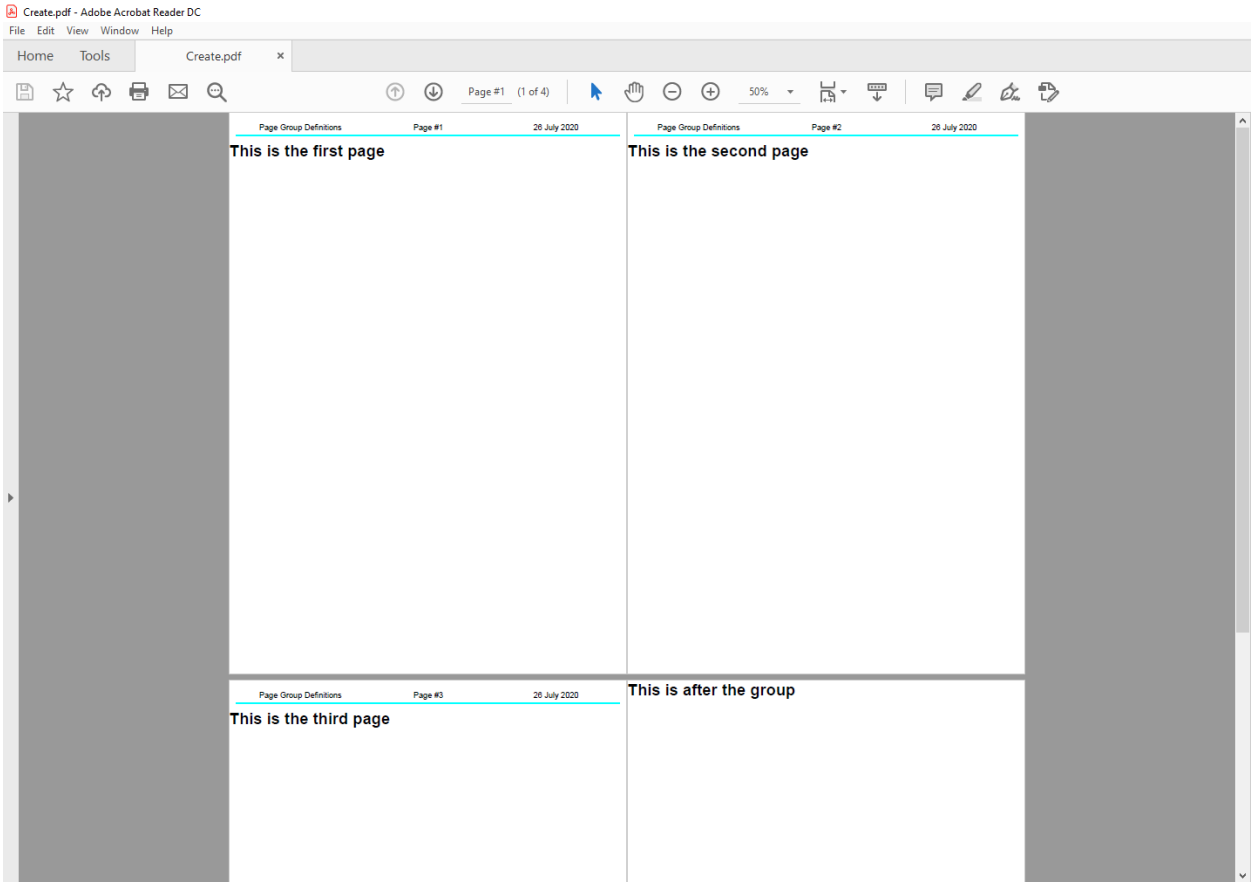
```
<?xml version="1.0" encoding="utf-8" ?>

<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
            xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
            xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd" >
<Styles>

    <!-- set up the default style for the header -->
    <styles:Style applied-type="pdf:Div" applied-class="header" >
        <styles:Border color="aqua" width="2" sides="Bottom"/>
        <styles:Padding all="5pt"/>
        <styles:Margins all="10pt"/>
        <styles:Font size="12pt"/>
        <styles:Position h-align="Center"/>
    </styles:Style>

    <!-- a page numbering style for the page groups -->
```

(continues on next page)

(continued from previous page)

```
    <styles:Style applied-type="pdf:PageGroup" >
        <styles:Page number-prefix="Page #" number-style="Decimals"/>
    </styles:Style>
</Styles>

<Pages>

    <pdf:PageGroup>
        <Params>
            <!-- Set parameters, just for this group -->
            <pdf:String-Param id="sectTitle" value="Page Group Definitions" ></
↪pdf:String-Param>
        </Params>
        <!-- consistent header across the pages in this group (split into 3 columns --
↪>
        <Header>
            <pdf:Div styles:class="header" styles:column-count="3" >
                <pdf:Label text="{@:sectTitle}" />
                <pdf:ColumnBreak/>
                <pdf:PageNumber />
                <pdf:ColumnBreak/>
                <pdf:Date styles:date-format="dd MMMM yyyy" />
            </pdf:Div>
        </Header>

        <Pages>

            <pdf:Page>
            <Content>
                <pdf:H3 >This is the first page</pdf:H3>
            </Content>
            </pdf:Page>

            <pdf:Section>
            <Content>
                <pdf:H3>This is the second page</pdf:H3>
                <pdf:PageBreak/>
                <pdf:H3>This is the third page</pdf:H3>
            </Content>
            </pdf:Section>

        </Pages>
    </pdf:PageGroup>

    <pdf:Page>
        <Content>
            <pdf:H3 >This is after the group</pdf:H3>
        </Content>
    </pdf:Page>

</Pages>

</pdf:Document>
```

By applying a header at the group level, we can be sure that it is repeated across all pages.



6.9.6 Page numbering

In the previous example we saw use of the reference/pdf_pagenumber component to display the current page number on a page. The actual numbering is held at a document level, but can be altered for each group, section or individual page.

See document_pagenumbers for a complete example.

6.10 Flowing pages and columns

Scryber supports flowing content layout. No matter the font, content type or structure.

It is as simple as html.

6.10.1 Putting content on a page

6.10.2 Flowing onto multiple pages

6.10.3 Breaking pages

6.10.4 Specifying columns

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.

6.10.5 Nested columns

6.10.6 Breaking columns

6.11 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.

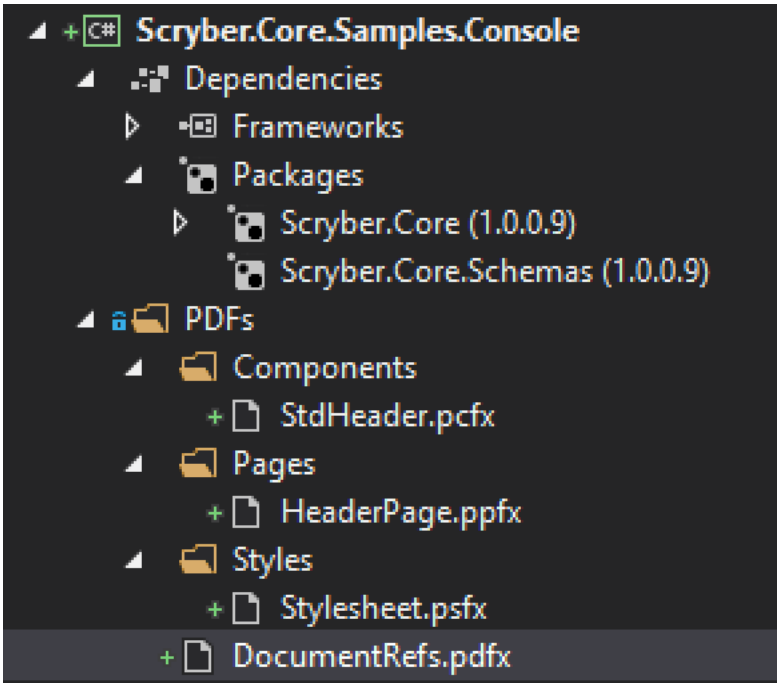
As a converions the files should have the following extensions.

- Documents - [MyDocument].pdfx
- Stylesheets - [MyStyles].psfx
- Pages - [MyPage].ppfx
- Components - [MyComponent].pcfx

It just makes life easier.

6.11.1 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.



6.11.2 DocumentRefs.pdfx

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

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
              xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd"
              auto-bind="true" >
    <Styles>
        <styles:Styles-Ref source="./Styles/Stylesheet.psfx"/>

    </Styles>

    <Pages>
        <pdf:Page-Ref source="Pages/HeaderPage.ppfx"></pdf:Page-Ref>

        <pdf:Page>
            <Header>
                <pdf:Component-Ref source="Components/StdHeader.pcfx"></pdf:Component-
↪Ref>
            </Header>
            <Content>
                <pdf:H1 styles:class="title" >This is the second Page </pdf:H1>
            </Content>
        </pdf:Page>
    </Pages>
</pdf:Document>
```

This contains a reference to *StyleSheet.psfx* in the *Styles* folder. A reference to a *HeaderPage.ppfx* in the *Pages* folder,

and a reference to a *StdHeader.pcfx* in the *Components* folder.

The path references are relative to the current document.

6.11.3 StyleSheet.psfx

This is the content of the *stylesheet.psfx*

```
<?xml version="1.0" encoding="utf-8" ?>
<styles:Styles xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
        xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Styles.
↪xsd">

    <styles:Style applied-class="title" >
        <styles:Font family="Segoe UI" bold="false" size="60pt"/>
        <styles:Margins all="20pt"/>
        <styles:Padding all="10pt"/>
        <styles:Position h-align="Center" />
    </styles:Style>


    <styles:Style applied-class="page-head" >
        <styles:Font family="Segoe UI" bold="false" size="14pt"/>
        <styles:Margins top="20pt" left="10pt" right="10pt" bottom="10pt" />
        <styles:Border sides="Bottom" style="Solid" width="1pt"/>
        <styles:Padding all="10pt"/>
    </styles:Style>

</styles:Styles>
```

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

6.11.4 HeaderPage.ppfx

This is the content of the *HeaderPage.psfx*, which has a reference to the *StdHeader.pcfx* file **relative** to the page, along with a heading.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Page xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
        xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Styles.
↪xsd">
    <Header>
        <pdf:Component-Ref source="../Components/StdHeader.pcfx"/>
    </Header>
    <Content>
        <pdf:H1 styles:class="title" text="Heading Page" ></pdf:H1>
    </Content>
</pdf:Page>
```

6.11.5 StdHeader.pcfx

The component is referenced from the *HeaderPage.ppfx* and also the *DocumentRefs.pdfx*. This file is just used as the content for the header of the pages.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Div xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
        xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Styles.
↪xsd"
        styles:class="page-head" styles:column-count="2" >

    <pdf:Label styles:class="head-text" text="Referenced Files Example" />
    <pdf:ColumnBreak/>
    <pdf:Date styles:class="head-text" styles:date-format="dd MMM yyyy" />
</pdf:Div>
```

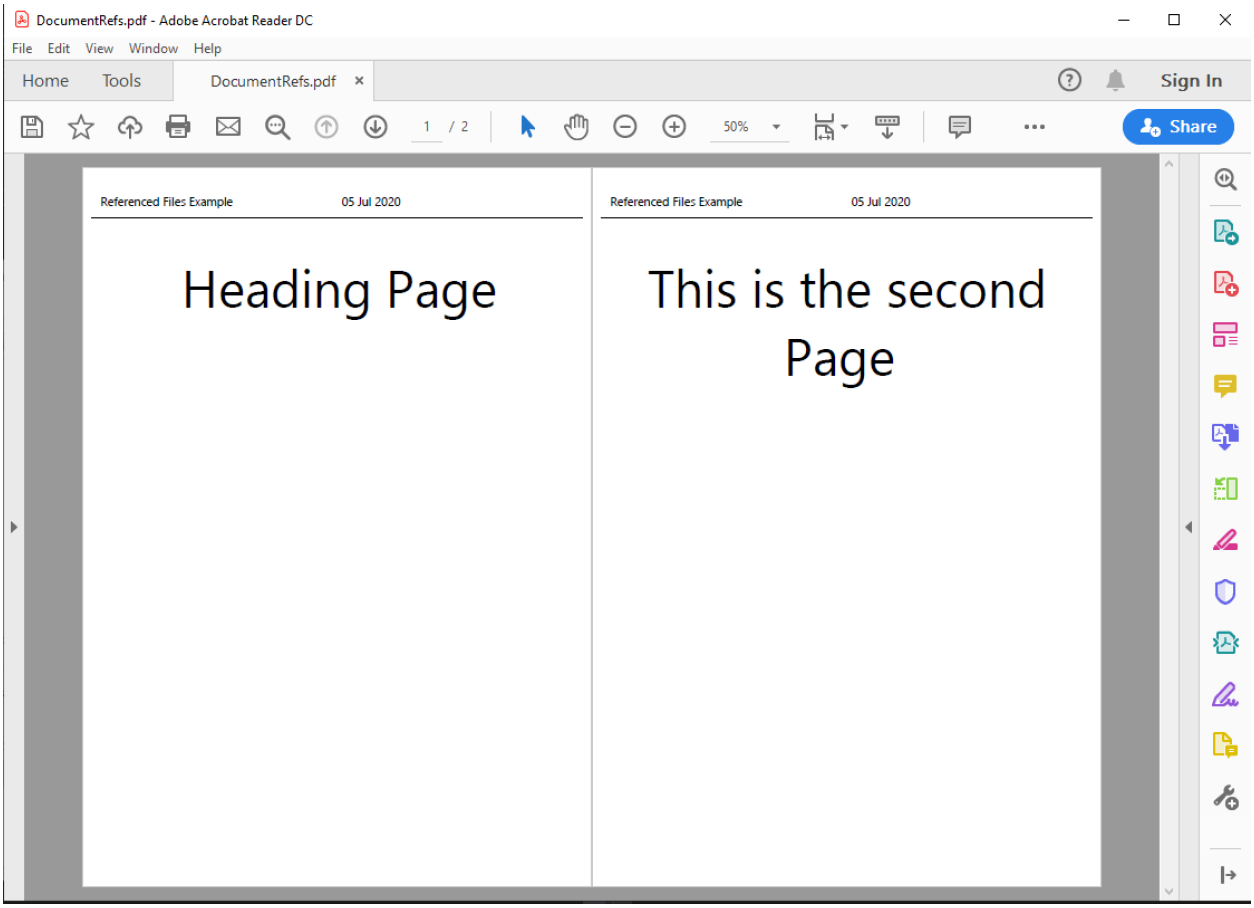
The content could be anything, but for this time we are using it as a standard header. It has 2 columns with a title on one side and then a date label on the other.

6.11.6 Bringing it all together

These are all the files, and we just need to generate them.

```
var path = System.IO.Path.Combine(workingDirectory, "PDFs", "DocumentRefs.pdfx");
using (var doc = PDFDocument.ParseDocument(path)){
    doc.ProcessDocument(outputPath, System.IO.FileMode.OpenOrCreate);
}
```

All being well, then when we bring it together we will get a 2 page document with consistent headers and content.



6.11.7 Overriding and passing data

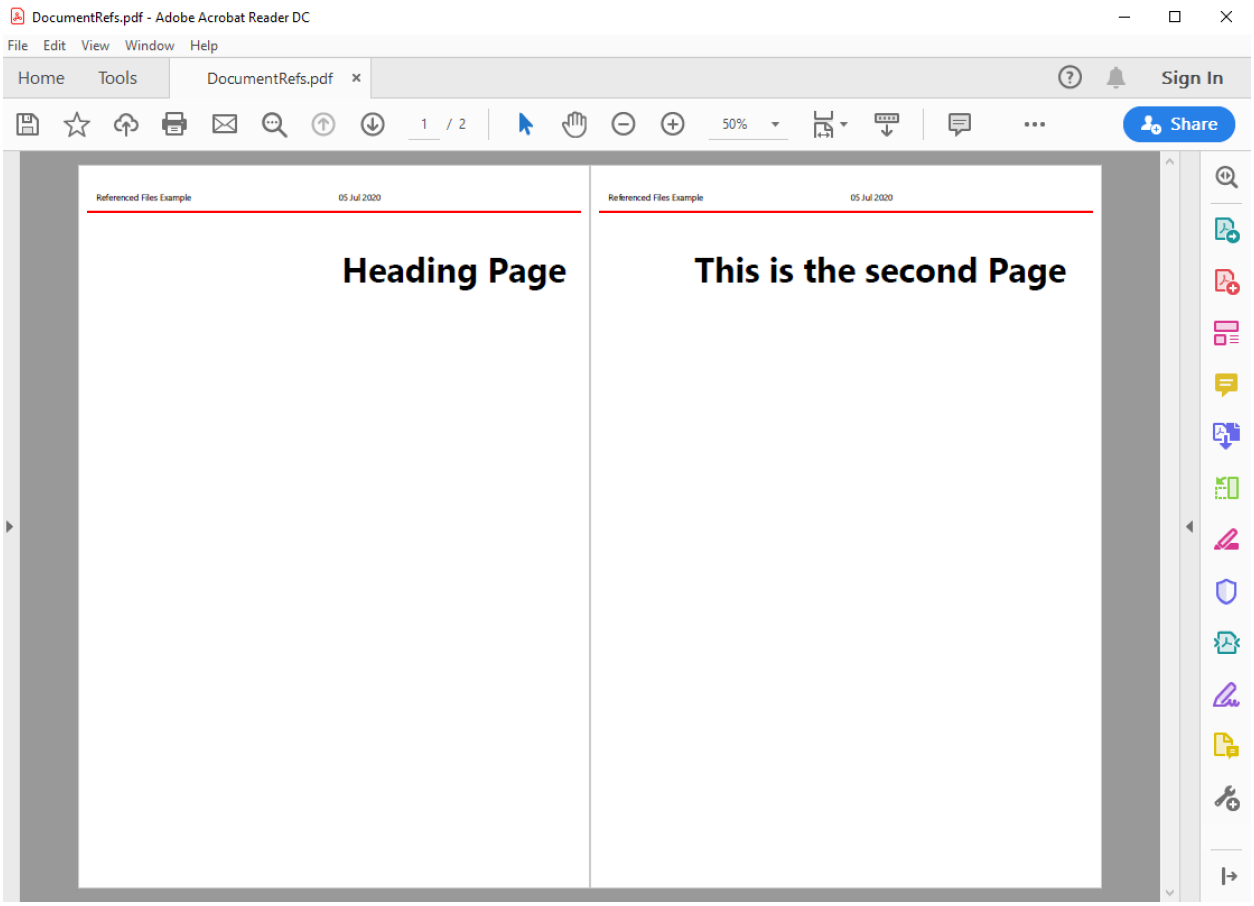
Using document styles and document parameters it is possible to modify the content of the document when it is bound.

To start with we can alter the styles that we have loaded from the style sheet.

```
   <Styles>
      <!-- Original Style sheet reference -->
      <styles:Styles-Ref source="./Styles/Stylesheet.psfx"/>

      <!-- Modification to the styles -->

      <styles:Style applied-class="title" >
          <styles:Font bold="true" size="40"/>
          <styles:Position h-align="Right"/>
      </styles:Style>


      <styles:Style applied-class="page-head" >
          <styles:Border color="red" width="2pt"/>
          <styles:Font size="10pt"/>
      </styles:Style>

</Styles>
```

These will be applied to the pages and components whenever they are referenced. Retaining the original properties

where they are unchanged.



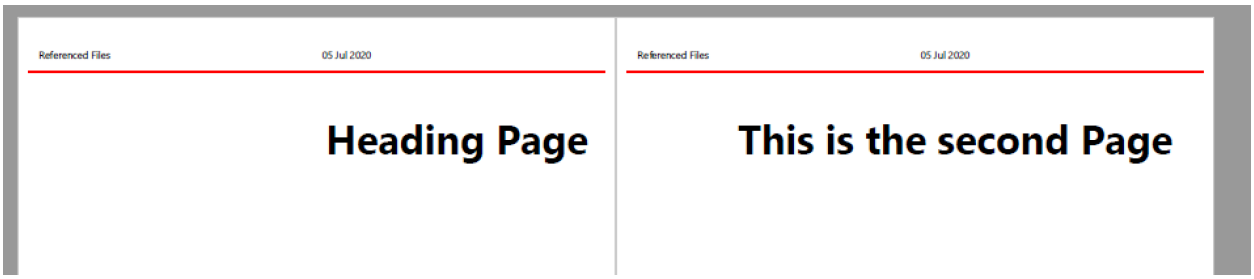
And then we can add parameters to our *DocumentRefs.pdfx* that we can use in our components and sub pages.

```
<Params>
    <pdf:String-Param id="doc-title" value="Referenced Files" />
</Params>
```

And reference that in our component *StdHeader.pcfx* with the standard binding notation on the text attribute **'{@:doc-title}'**

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Div xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
        xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Styles.
↪xsd"
        styles:class="page-head" styles:column-count="2" >

    <pdf:Label styles:class="head-text" text="{@:doc-title}" />
    <pdf:ColumnBreak/>
    <pdf:Date styles:class="head-text" styles:date-format="dd MMM yyyy" />
</pdf:Div>
```

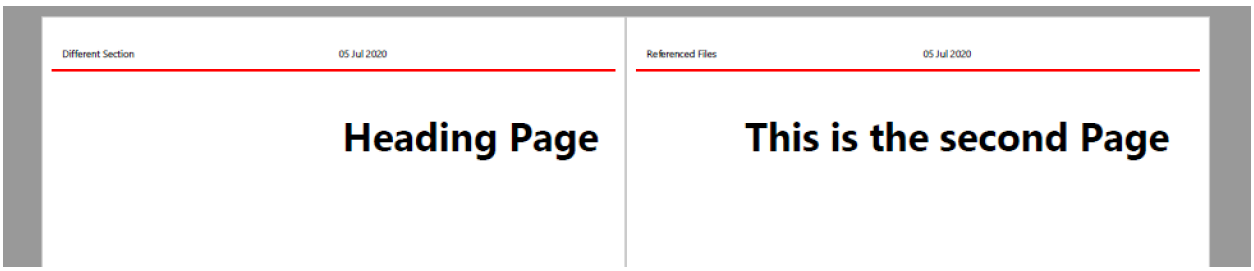
If we render this now, then the header should always use the *doc-title* value. If it is not provided, then it will simply be blank.



Finally we can put parameters explicitly in the template. These will only apply within the template and nowhere else. So we can provide a new value for the *doc-title* for our referenced page and that will be used on the header component, but it will revert back to the default value for our second actual page.

```
 <pdf:Page-Ref source="Pages/HeaderPage.ppfx">
  <Params>
    <pdf:String-Param id="doc-title" value="Different Section" />
  </Params>
</pdf:Page-Ref>
```

Rendering this will change the title for the header in the referenced component.



Note: You are not limited to strings in parameters, you can provide colours, data, xml and actual scryber components into the parameters.

Our full code for the *DocumentRefs.pdfx* file is

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
              xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd"
              auto-bind="true" >
<Styles>
    <styles:Styles-Ref source="./Styles/Stylesheet.psfx"/>

    <styles:Style applied-class="title" >
    <styles:Font bold="true" size="40"/>
    <styles:Position h-align="Right"/>
    </styles:Style>


    <styles:Style applied-class="page-head" >
    <styles:Border color="red" width="2pt"/>
    <styles:Font size="10pt"/>
```

(continues on next page)

(continued from previous page)

```
    </styles:Style>

</Styles>

<Params>
    <pdf:String-Param id="doc-title" value="Referenced Files" />
</Params>

<Pages>
    <pdf:Page-Ref source="Pages/HeaderPage.ppfx">
    <Params>
        <pdf:String-Param id="doc-title" value="Different Section" />
    </Params>
    </pdf:Page-Ref>

    <pdf:Page>
    <Header>
        <pdf:Component-Ref source="Components/StdHeader.pcfx"></pdf:Component-Ref>
    </Header>
    <Content>
        <pdf:H1 styles:class="title" >This is the second Page </pdf:H1>
    </Content>
    </pdf:Page>
</Pages>
</pdf:Document>
```

6.11.8 Circular references

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

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

6.11.9 Selecting within a file

Because we use XML as the native store for the files we can also use XPath to select specific components within a file. If you wanted to pull out just a heading from a file with id *title* you could use:

```
<pdf:Component-Ref source='Components/StdHeader.pcfx' select='//pdf:Div/pdf:H1[@id=
↪"title"]' />
```

This would then only load that component, and not any other components in that file. It's quite useful to build a library of standard components all together without creating a plethora of files.

6.12 Document drawing units and measures - td

There are a number of positioning and sizing structures in scryber. All based around the Unit of measure.

- PDFUnit
- PDFSize

- PDFPoint
- PDFThickness
- PDFRect

6.13 Document colours, fills and backgrounds - td

There are a number of positioning and sizing structures in scryber. All based around the Unit of measure.

- PDFUnit
- PDFSize
- PDFPoint
- PDFThickness
- PDFRect

6.14 Images in documents - td

Content to come

6.15 Document fonts and font styles - td

Fonts are by detault automatically loaded at startup, and can be referenced by name. Scryber supports the following font file format

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

6.16 Drawing paths and shapes - td

Scryber includes a full drawing capability.

- Lines
- Recangles
- Elipses
- Bezier Curves
- Groups
- etc.

6.17 A Scryber Document outline - td

Content to complete

6.18 Passing data to your template

A document template is just that, a template. In your code you can add any source of information to be included.

- As an value or model for the parameters
- Bound to an XML datasource (xml, sql or object)
- As a Controller with an event method
- Just in code

And they can be used in your document with many of the data controls

- The document itself
- A Data Grid
- A Data List
- *With* a single entry
- *ForEach* loop

And the content supports as default both object and xpath binding. The notation for an binding on an attribute is based on the { and } with a method (@ or xpath for the built in binders), a colon ':', and then finally the selector.

e.g. *attribute='{@:paramName}'* for objects or *attribute='{xpath:selector}'* for xml

6.18.1 The Document parameters

Every Document can have parameters associated with it. These should be declared at the top of the Document, in the Params element, for clarity to other developers (even if the default value is empty).

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

<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
            xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
            xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd" >
<Params>
    <!-- Declare a complex object parameter -->
    <pdf:Object-Param id="Model" />
</Params>

<Pages>
    <!-- Use the models 'DocTitle' property for the outline. -->
    <pdf:Page outline-title="{@:Model.DocTitle}" styles:margins="20pt">
    <Content>
        <!-- And use it as the text on the heading -->
        <pdf:H1 styles:class="title" text="{@:Model.DocTitle}" > </pdf:H1>

        <pdf:Ul>
        <!-- now we loop through the 'Entries' property -->
        <data:ForEach value="{@:Model.Entries}" >
            <Template>
            <pdf:Li>
                <!-- and create a list item for each entry (. prefix) with the name␣
↪property. -->
```

(continues on next page)

(continued from previous page)

```
                <pdf:Text value="{@:.Name}" />
            </pdf:Li>
            </Template>
        </data:ForEach>
        </pdf:Ul>

    </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```

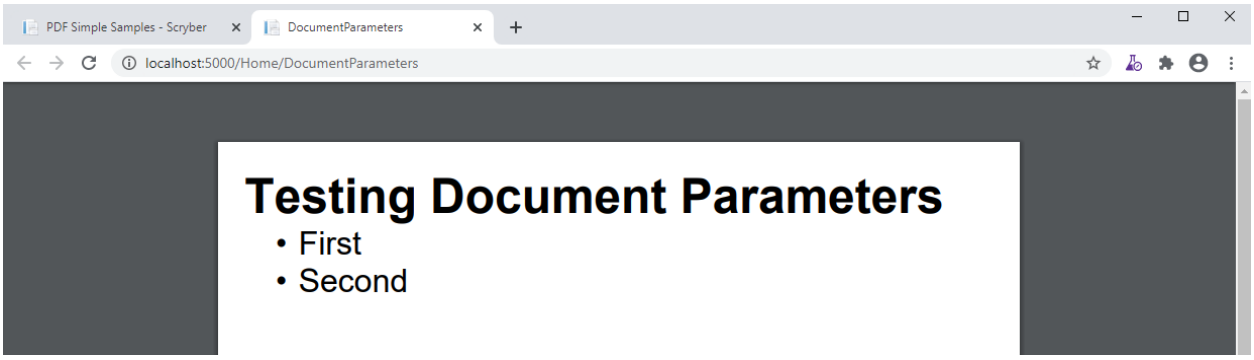
And the value can be set or changed at runtime

```
var model = new {
    DocTitle = "Testing Document Parameters",
    Entries = new[] {
            new { Name = "First", Id = "FirstID"},
            new { Name = "Second", Id = "SecondID" }
        }
};

var pdfDoc = PDFDocument.ParseDocument(path);
pdfDoc.Params["Model"] = model;

pdfDoc.ProcessDocument(output);
```

Or passed as the Model in the MVC methods

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

    // This could be any object dynamically built or strongly typed.
    var model = new
    {
        DocTitle = "Testing Document Parameters",
        Entries = new[] {
                new { Name = "First", Id = "FirstID"},
                new { Name = "Second", Id = "SecondID" }
            }
    };

    //This method always stores the passed model as the `Model` parameter
    return this.PDF(path, model);
}
```

And this will be used in the output.



See *Parameters and object binding* for full details.

6.18.2 The Datasources

Putting the document more in control of the data it uses, is supported from the available DataSources and Commands that sit in the *Data* element of the document.

This element should contain all the datasources required by the document. They can be an XML file, or XML Http request, a SQL database call, an object call, or a json request

e.g. This document has an xml content reference from a remote source (in this case a local host controller method). That returns the following content..

```
<?xml version="1.0" encoding="utf-8" ?>
<DataSources title="Testing Xml Datasources">
    <Entries>
        <Entry Name="First Xml" Id="FirstID" />
        <Entry Name="Second Xml" Id="SecondID" />
    </Entries>
</DataSources>
```

And with that we can bind the source into the document

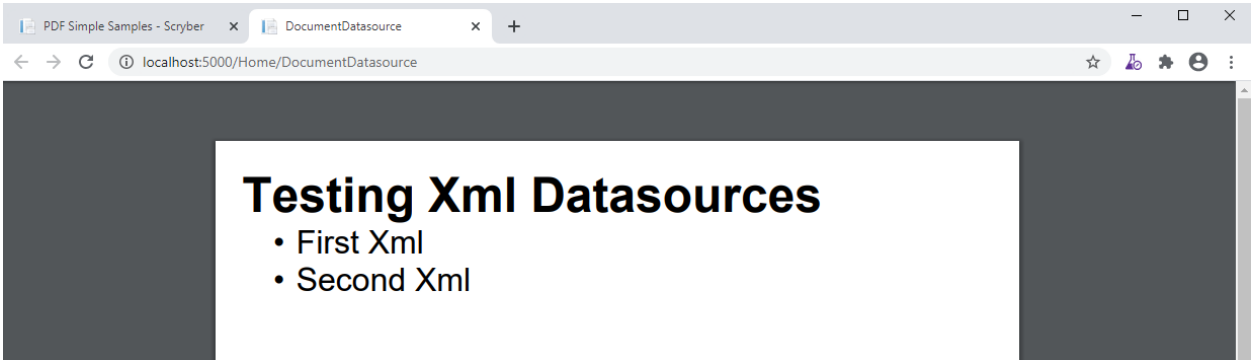
```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
            xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
            xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd" >
    <Data>
        <!-- This is a data source declared witin the document, that pulls the xml␣
↪from the feed -->
        <data:XMLDataSource id="XmlSource" source-path="http://localhost:5000/Home/Xml
↪" ></data:XMLDataSource>
    </Data>
    <Pages>

        <pdf:Page styles:margins="20pt">
        <Content>
            <!-- Use the `data:With` component to specify a source and path within␣
↪the xml as a starting point. -->
            <data:With datasource-id="XmlSource" select="//DataSources" >
```

(continues on next page)

(continued from previous page)

```
            <!-- And use it as the text on the heading -->
            <pdf:H1 styles:class="title" text="{xpath:@title}" > </pdf:H1>

            <pdf:Ul>
                <!-- now we loop through the 'Entries' property -->
                <data:ForEach value="{xpath:Entries/Entry}" >
                <Template>
                    <pdf:Li>
                    <!-- and create a list item for each entry (. prefix) with the␣
↪name property. -->
                    <pdf:Text value="{xpath:@Name}" />
                    </pdf:Li>
                </Template>
                </data:ForEach>
            </pdf:Ul>
            </data:With>

        </Content>
        </pdf:Page>
    </Pages>

</pdf:Document>
```

With the result of the output showing the content.



We could have specified the source on the *data:ForEach*, and alternatively we could have used a Json DataSource to return an object binding. See *Data Binding in Scryber - td* for more details.

6.18.3 The Document Controller

The most complex, but utlimately most adaptable is specifying a controller class on your template

The document file or referenced files have Controllers associated with them to handle events and properties. This gives complete control back to your code during the lifecycle of the document.

It is based on providing a type on the scryber processing instruction

```
<?scryber controller='Full.Type.Name, Assembly.Name' ?>
```

Note: More details of the scryber processing instruction can be found in the :doc:document_structure document.

A full document file example is below.

```
<?xml version="1.0" encoding="utf-8" ?>
<?scryber controller='Scryber.Core.Samples.Web.Controllers.DocumentControllerInstance,
↪ Scryber.Core.Samples.Web' ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
            xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd"
            xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd"
            on-loaded="LoadDocument" >
<Pages>

    <pdf:Page styles:margins="20pt">
    <Content>

        <!-- This will automatically be set on the controller instance property -->
        <pdf:H1 id="Title" > </pdf:H1>

        <pdf:Ul>
            <!-- now we call the BindForEach method to set the data value -->
            <data:ForEach on-databinding="BindingForEach" >
            <Template>
                <!-- and finally we use the item data bound to set the
                    content of the list item for each entry -->
                <pdf:Li on-databound="BoundListItem"></pdf:Li>
            </Template>
            </data:ForEach>
        </pdf:Ul>

    </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```

The document has declared:

- The on-loaded event for LoadDocument.
- It has a heading with ID of Title.
- A ForEach with a databinding handler
- And a List item inside the template which has binding mapped to another handler.

In our controller we declare explicitly our outlets (properties) and actions (methods).

```
namespace Scryber.Core.Samples.Web.Controllers
{
    public class DocumentControllerInstance
    {
        /// <summary>
        /// The Heading will be set on a controller instance from the parser
        /// </summary>
        [PDFOutlet()]
        public PDFHead1 Title
        {
            get;set;
        }
```

(continues on next page)

(continued from previous page)

```
        /// <summary>
        /// Parameterless constructor
        /// </summary>
        public DocumentControllerInstance()
        {
        }


        [PDFAction()]
        public void LoadDocument(object sender, PDFLoadEventArgs args)
        {
            //Document loaded, so set the title text
            this.Title.Text = "Test Controller Title";
        }

        //Just some sample data.
        string[] data = new[] { "First", "Second", "Third" };

        /// <summary>
        /// Happens just before the ForEach is DataBound, so that we can assign the␣
↪data value, and that will be used.
        /// </summary>
        [PDFAction()]
        public void BindingForEach(object sender, PDFDataBindEventArgs args)
        {
            //Dynamically set the data on the ForEach component - so it will loop␣
↪through
            var forEach = (Data.PDFForEach)sender;
            forEach.Value = data;
        }

        /// <summary>
        /// Happens 3 times for each of the list items created in the template from␣
↪the data source.
        /// </summary>
        [PDFAction()]
        public void BoundListItem(object sender, PDFDataBindEventArgs args)
        {
            var listItem = (PDFListItem)sender;
            var index = args.Context.CurrentIndex;
            var text = data[index];
            //Create a new text literal and add it to the listitem
            PDFTextLiteral literal = new PDFTextLiteral(text);
            listItem.Contents.Add(literal);
        }
    }
}
```

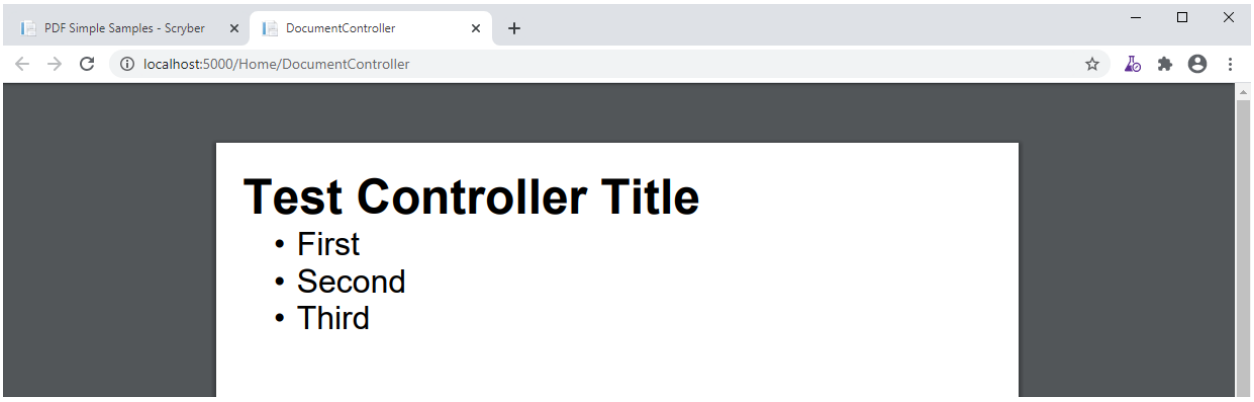
Generating the file is exaclty the same process but the parser will discover the controller class, apply the outlets and actions, and then execute. The result should come out with the content dynamically assigned.

```
public IActionResult DocumentController()
{
    var path = _rootPath;
    path = System.IO.Path.Combine(path, "Views", "PDF", "DocumentController.pdfx");
```

(continues on next page)

(continued from previous page)

```
    return this.PDF(path);
}
```



For more information on controllers and the event model see *Using controllers with your templates - td* and *Lifecycle of a document creation - td*

6.18.4 Which should I use?

All 3 methods of generating dynamic content within your template have their own benefits, and they are not mutually exclusive.

- **The simplest is using parameters but the model can become too complex.**
  - *Parameters and object binding*
- **Moving the model to one or more data sources can be a quick solution as complexity increases**
  - *Data Binding in Scryber - td*
- **Adding a controller gives complete 'control' for complex business logic.**
  - *Using controllers with your templates - td*

6.19 Parameters and object binding

The document parameters are values that can be set within the template or in code. They are able to hold values used through out the generation process (see *Lifecycle of a document creation - td*) and are passed via the Context to any event handlers.

6.19.1 Declaring and Using in Documents

Document parameters are declared within the <*Params*> element of Declaring a parameter in a document is not required for it to be used later on, but is best practice. It is more readable and supports default values if they do not change.

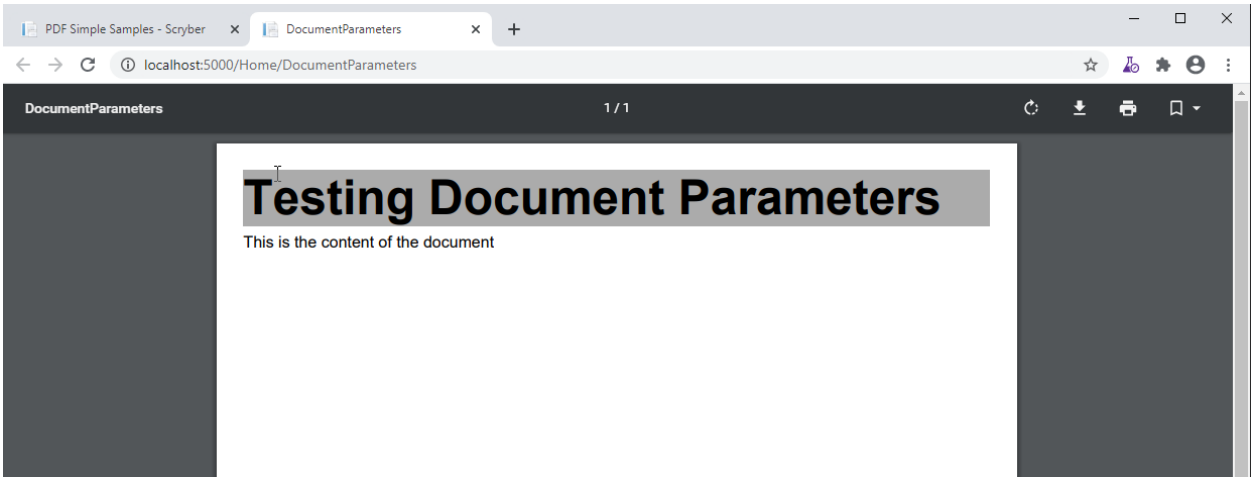
When referring to a parameter within your document template use the @: binding syntax (or item: syntax) e.g. *{@:MyParamName}* Then the parser finds a binding reference with this syntax it will create a binding expression that will be evaulated in the *Lifecycle of a document creation - td* Databinding phase and the value applied to the property it was set on.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd" >
    <Params>
        <!-- Declare the parameters -->
        <pdf:Bool-Param id="ShowTitle" value="true" />
        <pdf:String-Param id="MyTitle" value="Document Title" />
        <pdf:Color-Param id="TitleBg" value="#AAAAAA" />
    </Params>

    <Pages>
        <!-- Use the 'MyTitle' parameter for the outline. -->
        <pdf:Page outline-title="{@:MyTitle}" styles:margins="20pt" styles:font-size=
↪"12pt">
        <Content>
            <!-- And use it as the text on the heading -->
            <pdf:H1 visible="{@:ShowTitle}" styles:bg-color="{@:TitleBg}"  text="
↪{@:MyTitle}" > </pdf:H1>
            <pdf:Para >This is the content of the document</pdf:Para>
        </Content>
        </pdf:Page>
    </Pages>

</pdf:Document>
```

Generating this document the parameter values are applied to the final output and rendered.

```
[HttpGet]
public IActionResult DocumentParameters()
{
    var path = _rootPath;
    path = System.IO.Path.Combine(path,"Views", "PDF", "DocumentParameters.pdfx");
    var doc = PDFDocument.ParseDocument(path);

    return this.PDF(doc);
}
```

And this then output as follows

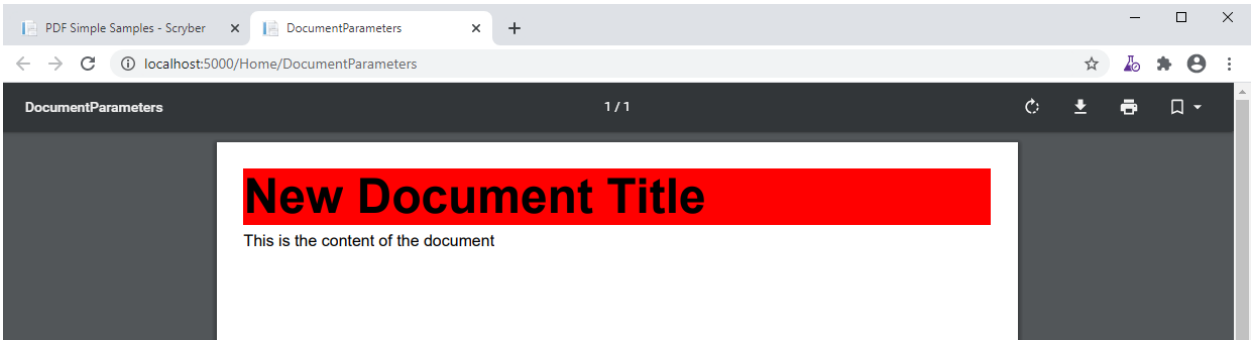


6.19.2 Changing the values

Parameters are evaluated during the data binding stage of a document creation, so once bound, any changes to the parameters will not be evaluated. The best time to change the values of a parameters are once it has been parsed, or on the loaded event.

```
[HttpGet]
public IActionResult DocumentParameters()
{
    var path = _rootPath;
    path = System.IO.Path.Combine(path,"Views", "PDF", "DocumentParameters.pdfx");
    var doc = PDFDocument.ParseDocument(path);

    doc.Params["MyTitle"] = "New Document Title";
    doc.Params["TitleBg"] = new PDFColor(255,0,0);

    return this.PDF(doc);
}
```

Generating the file will then apply the vaules at binding time to the content and rendering will produce the following output.



It is perfectly acceptable to assign a parameter in the document that is not declared, nor does it have to be typed.

e.g.

```
[HttpGet]
public IActionResult DocumentParameters()
{
    var path = _rootPath;
    path = System.IO.Path.Combine(path,"Views", "PDF", "DocumentParameters.pdfx");
    var doc = PDFDocument.ParseDocument(path);

    doc.Params["MyTitle"] = "New Document Title";
    doc.Params["TitleBg"] = new PDFColor(255,0,0);
    //Undeclared parameter
    doc.Params["Size"] = (PDFUnit)30;
    return this.PDF(doc);
}
```

And the used in your template

```
<pdf:H1 visible="{@:ShowTitle}" styles:font-size="{@:Size}" styles:bg-color="
↪{@:TitleBg}"  text="{@:MyTitle}" > </pdf:H1>
```

But it will not be co-erced into the correct type, nor will it have a clear initial value.

6.19.3 Simple Parameter Types

Scryber is strongly typed. The xml templates are defined as classes in namespaces and assemblies, and so are the **parameter** declarations.

There are a range of types available, and options for using complex types (see below).

- String-Param: Any string value, the default if not set is null.
- Int-Param: Single integer value, the defult if not set is 0.
- Guid-Param: A GUID value, the default is an empty guid.
- Double-Param: Holds double values, the default is 0.0
- Bool-Param: Boolean (True, False) values, the default is false.
- Date-Param: Date and time values, the default is minimum date time and values are culture sensitive.
- Unit-Param: Holds a reference/pdf_unit value, see *Positioning your content* for more info. The default is empty (zero) unit.
- Color-Param: Holds a reference/pdf_color value, the default is transparent.
- Thickness-Param: Holds a reference/pdf_thickness value (used in padding, margins, clipping etc.). The default is empty (zero) thickness.
- Enum-Param: Has a specific *type* attribute that specifies the type of enum that should be stored. The default is null.

There are 3 other parameter types available XML, Template and Object which are discussed later on in this document.

6.19.4 Complex Object Parameters

Whilst Scryber Parameters can be simple types, it also supports complex objects that can be traversed.

Our previous example could have been written with a single parameter rather than the 3 individual ones, an the values retrieved from the properties on that object.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
            xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd" >
    <Params>
        <!-- Declare the parameters -->
        <pdf:Object-Param id="Heading" />
    </Params>

    <Pages>
        <!-- Use the 'MyTitle' parameter for the outline. -->
        <pdf:Page outline-title="{@:Heading.Title}" styles:margins="20pt" styles:font-
↪size="12pt">
        <Content>
            <!-- And use it as the text on the heading -->
            <pdf:H1 visible="{@:Heading.Visible}" styles:bg-color="{@:Heading.
↪Background}"  text="{@:Heading.Title}" > </pdf:H1>
            <pdf:Para >This is the content of the document</pdf:Para>
        </Content>
        </pdf:Page>
    </Pages>
```

(continues on next page)

(continued from previous page)

```
</pdf:Document>
```

The dot notation is evaluated at runtime to bind the appropriate value.

```
[HttpGet]
public IActionResult DocumentParameters()
{
    var path = _rootPath;
    path = System.IO.Path.Combine(path,"Views", "PDF", "DocumentParameters.pdfx");
    var doc = PDFDocument.ParseDocument(path);

    //Set the heading param to a new dynamic type.
    doc.Params["Heading"] = new
    {
        Title = "Model Document Title",
        Visible = true,
        Background = "#FF0000"
    };

    return this.PDF(doc);
}
```

It is also possible to stronly type the object parameter by specifying the expected **full** type name, so you can be sure the content coming into the template matches. Inherited types will be acceptable as will interfaces.

```
<Params>
    <!-- Declare the parameters -->
    <pdf:Object-Param id="Heading" type="MyNamespace.MyType, MyAssembly" />
</Params>
```

```
doc.Params["Heading"] = new MyNamespace.MyType("Title",true, "#FF0000");
```

Note: If you provide a class that is not assignable to the parameter type a PDFDataException will be raised directly on assignment, so easily troubleshooted.

6.19.5 The MVC model

In the scryber.core.mvc project, there is a special extension method on the controller that accepts not just the document, but also an object as the model. Within this extension method, the *Model* parameter value will directly be assigned, even if it does not exist.

```
[HttpGet]
public IActionResult DocumentParameters()
{
    var path = _rootPath;
    path = System.IO.Path.Combine(path,"Views", "PDF", "DocumentParameters.pdfx");
    var doc = PDFDocument.ParseDocument(path);

    //Set the heading param to a new dynamic type.
    var model = new MyNamespace.MyType("Title",true, "#FF0000");
```

(continues on next page)

(continued from previous page)

```
    return this.PDF(doc, model);
}
```

And in your template, you can specify the model type you are expecting.

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
            xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Styles.xsd" >
    <Params>
        <!-- Declare the parameters -->
        <pdf:Object-Param id="Model" type="MyNamespace.MyType, MyAssembly" />
    </Params>

    <Pages>
        <!-- Use the 'MyTitle' parameter for the outline. -->
        <pdf:Page outline-title="{@:Model.Title}" styles:margins="20pt" styles:font-
↪size="12pt">
        <Content>
            <!-- And use it as the text on the heading -->
            <pdf:H1 visible="{@:Model.Visible}" styles:bg-color="{@:Model.Background}
↪"  text="{@:Model.Title}" > </pdf:H1>
            <pdf:Para >This is the content of the document</pdf:Para>
        </Content>
        </pdf:Page>
    </Pages>

</pdf:Document>
```

6.19.6 Combining selector paths

The object selectors support complex notation for retrieving values.

- **{@:*dotnotation*} for binding to a paramter passed to the document. This supports complex paths**
  - {@:ParamName} for the direct value.
  - {@:ParamName.Property} for getting a property value.
  - {@:ParamName[n]} for getting the n'th value from an array
  - {@:ParamName['key']} for geting a dictionary value based on key.
- **The statements can be chained together as long as needed.**
  - {@:Model.Property[0].Property['key'].Value}
  - If one of the properties evaluates to null, then the chain will no longer be evaluated, and no value will be set.

6.19.7 Binding to Collections

With complex objects it is possible to bind to object arrays or any other type of collection. The object that is extracted from the collection at that time will become the current *context*.

To refer to the properties in the current context simply precede the property with a dot (.)

```
<?xml version='1.0' encoding='utf-8' ?>
    <pdf:Document xmlns:pdf = 'http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Components.xsd'
                  xmlns:styles = 'http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd'
                  xmlns:data = 'http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Data.xsd'
                      >
    <Params>
        <pdf:Object-Param id='Model' ></pdf:Object-Param>
    </Params>

    <Pages>

    <pdf:Section>
        <Content>

            <data:ForEach value='{@:Model.List}' >
                <Template>
                    <!-- Here we can refer to the current object and set values from␣
↪properties. -->
                    <pdf:Label id='{@:.Id}' text='{@:.Name}' ></pdf:Label>
                    <pdf:Br/>
                </Template>
            </data:ForEach>

        </Content>
    </pdf:Section>

    </Pages>
</pdf:Document>
```

And when we are providing the value we can add an array or list.

```
doc.Params["Model"] = new
{
    Color = Scryber.Drawing.PDFColors.Aqua,
    List = new[] {
        new { Name = "First", Id = "FirstID"},
        new { Name = "Second", Id = "SecondID" }
    }
};
```

For more on looping through content and the available data components see *Data Binding in Scryber - td*

6.19.8 Binding Styles to Parameters

As styles are full qualified members of the document object, they also support databinding to values.

```
<?xml version='1.0' encoding='utf-8' ?>
<pdf:Document xmlns:pdf = 'http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd'
                  xmlns:styles = 'http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd'
                  xmlns:data = 'http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Data.xsd' >
```

(continues on next page)

(continued from previous page)

```
<Params>
    <pdf:Object-Param id='Model' ></pdf:Object-Param>
</Params>

<Styles>
    <!-- Bind the head and body styles to the Theme -->
    <styles:Style applied-class='head'>
    <styles:Padding all='20pt'/>
    <styles:Background color='{@:Model.Theme.TitleBg}' />
    <styles:Fill color='{@:Model.Theme.TitleColor}'/>
    <styles:Font family='{@:Model.Theme.TitleFont}' bold='false' italic='false' />
    </styles:Style>

    <styles:Style applied-class='body'>
    <styles:Font family='{@:Model.Theme.BodyFont}' size='{@:Model.Theme.BodySize}' />
    <styles:Fill color='#333300'/>
    <styles:Padding all='20pt'/>
    </styles:Style>

</Styles>

<Pages>

    <pdf:Section>
    <Content>
        <!-- Specify the class names on the components to use the styles -->
        <pdf:H1 styles:class='head' text='{@:Model.Title}' ></pdf:H1>
        <pdf:Div styles:class='body' >
        <!-- and then loop through -->
        <data:ForEach value='{@:Model.List}' >
            <Template>
            <pdf:Label id='{@:.Id}' text='{@:.Name}' ></pdf:Label>
            <pdf:Br/>
            </Template>
        </data:ForEach>
        </pdf:Div>
    </Content>
    </pdf:Section>

</Pages>
</pdf:Document>
```

And we can generate this content by providing the Theme as well as the List.

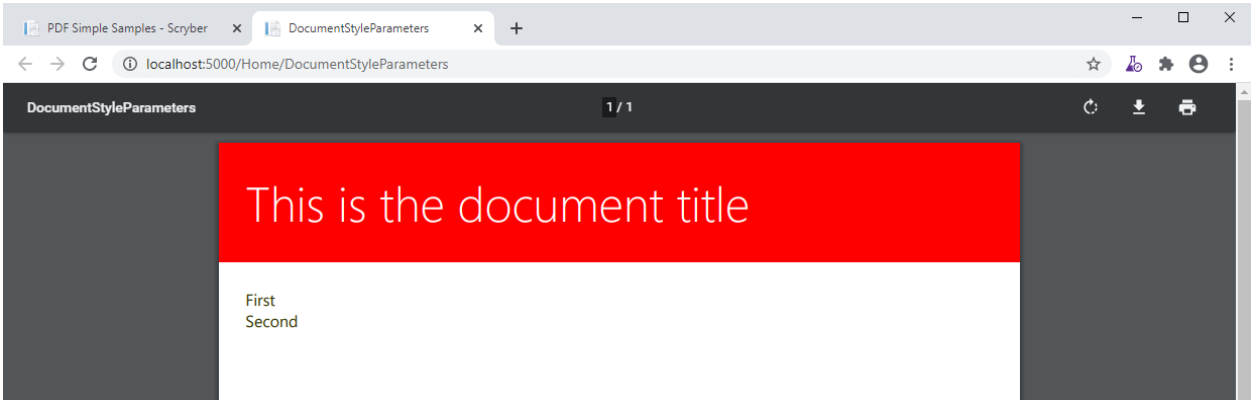
```
[HttpGet]
public IActionResult DocumentStyleParameters()
{
    var path = _rootPath;
    path = System.IO.Path.Combine(path, "Views", "PDF", "DocumentStyleParameters.pdfx
↪");
    var doc = PDFDocument.ParseDocument(path);

    var model = new
    {
        Title = "This is the document title",
        List = new[] {
            new { Name = "First", Id = "FirstID" },
```

(continues on next page)

(continued from previous page)

```
            new { Name = "Second", Id = "SecondID" }
        },
        Theme = new
        {
            TitleBg = new PDFColor(1,0,0),
            TitleColor = new PDFColor(1,1,1),
            TitleFont = "Segoe UI Light",
            BodyFont = "Segoe UI",
            BodySize = (PDFUnit)12
        }
    };

    return this.PDF(doc, model);
}
```

These styles should then be used in the creation of the document



Very quickly our document complexity can grow and then it becomes more important to split the data from the content, and we can do that using the document_datasources and *Using controllers with your templates - td*

6.19.9 XML parameters

Along with the object parameters, scryber supports the use of XML as a parameter. These are just as poweful as objects.

The xml data parameter, similar to the object parameter supports full xpath deep binding, and functions such as substring and concat. For more details on the xpath syntax see the document_datasources

```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd"
             xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd">
<Params>
    <!-- Declare the parameters -->
    <pdf:String-Param id="MyTitle" value="Document Title" />

    <!-- This is the xml content that will be used by default -->
    <pdf:Xml-Param id="MyData" >
```

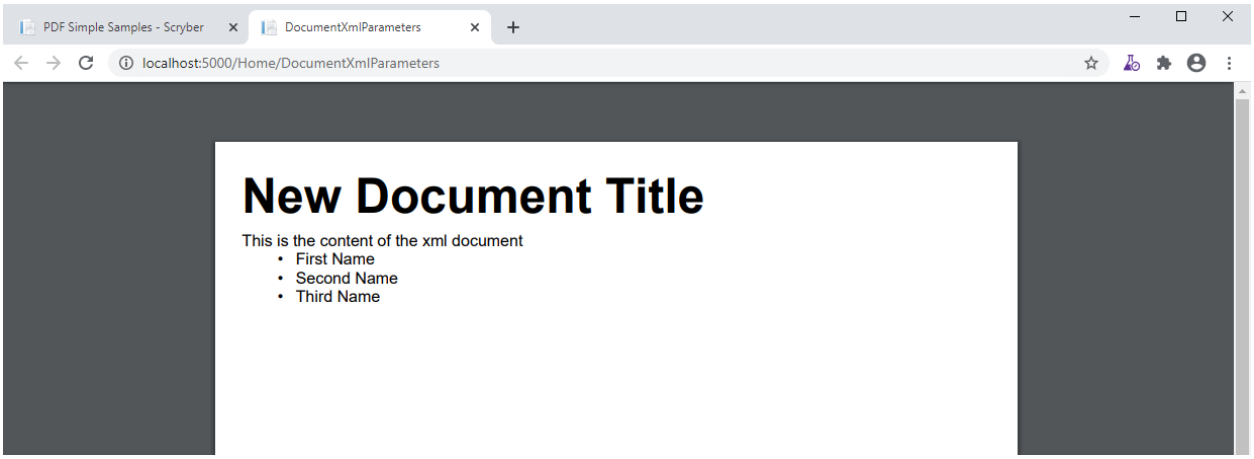
(continues on next page)

(continued from previous page)

```
    <Root>
        <Entry id="First">First Name</Entry>
        <Entry id="Second">Second Name</Entry>
        <Entry id="Third">Third Name</Entry>
    </Root>
    </pdf:Xml-Param>

</Params>

<Pages>
    <!-- Use the 'MyTitle' parameter for the outline. -->
    <pdf:Page outline-title="{@:MyTitle}" styles:margins="20pt" styles:font-size="12pt
↪">
    <Content>
        <!-- And use it as the text on the heading with a visble flag and background -
↪->
        <pdf:H1  text="{@:MyTitle}" > </pdf:H1>
        <pdf:Para >This is the content of the xml document</pdf:Para>

        <pdf:Ul>
            <!-- Now bind the content of the MyData parameter into a foreach, with␣
↪the selector of //Root/Entry
                to loop through each one in turn -->
            <data:ForEach value="{@:MyData}" select="//Root/Entry" >
            <Template>
                <pdf:Li >
                <pdf:Text value="{xpath:text()}" />
                </pdf:Li>
            </Template>
            </data:ForEach>
        </pdf:Ul>

    </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```

If we generate this content as is the xml will be bound to the unordered list and created.

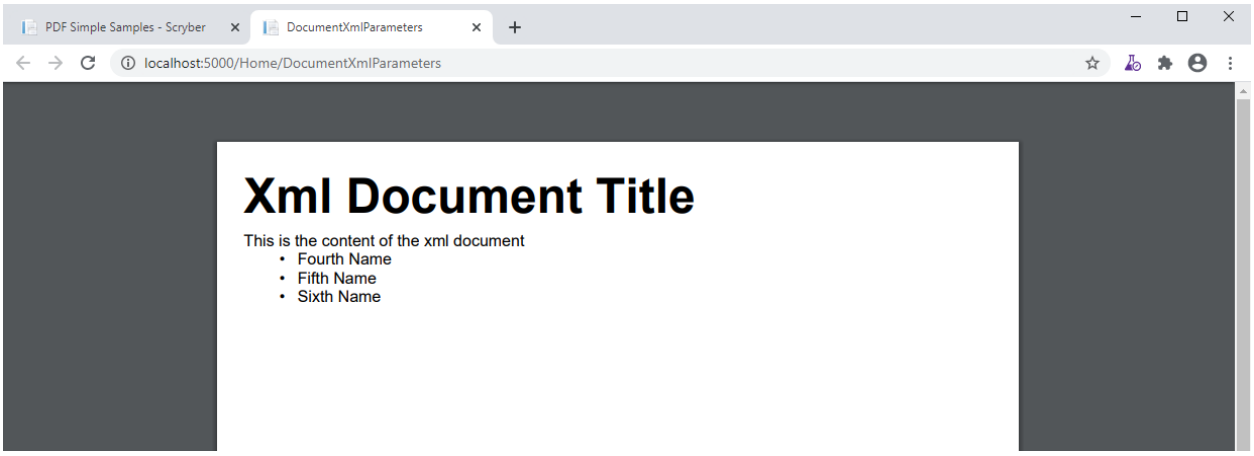
```
[HttpGet]
public IActionResult DocumentXmlParameters()
{
    var path = _rootPath;
    path = System.IO.Path.Combine(path, "Views", "PDF", "DocumentXmlParameters.pdfx");
    var doc = PDFDocument.ParseDocument(path);

    doc.Params["MyTitle"] = "New Document Title";

    return this.PDF(doc);
}
```



By using the xml data as a template we can generate this dynamically too, or load it from a file, or pull from a service. The xml parameter will accept XmlNode values, XPathNavigators, and Linq XElements for values, along with strings.

```
[HttpGet]
public IActionResult DocumentXmlParameters()
{
    var path = _rootPath;
    path = System.IO.Path.Combine(path, "Views", "PDF", "DocumentXmlParameters.pdfx");
    var doc = PDFDocument.ParseDocument(path);

    doc.Params["MyTitle"] = "Xml Document Title";

    //Replace the xml content in the MyData parameter
    var ele = new XElement("Root",
        new XElement("Entry", new XAttribute("id", "Fourth"), new XText("Fourth Name
↪")),
        new XElement("Entry", new XAttribute("id", "Fifth"), new XText("Fifth Name")),
        new XElement("Entry", new XAttribute("id", "Sixth"), new XText("Sixth Name"))
        );
    doc.Params["MyData"] = ele;

    return this.PDF(doc);
}
```

Generating this file again will render the content with the new xml data.



6.19.10 Template Parameters

Along with the XML parameter, scryber supports the Template parameter, which is xml content of scryber components. So you can provide both dynamic data, and dynamic structure to your document at generation time.

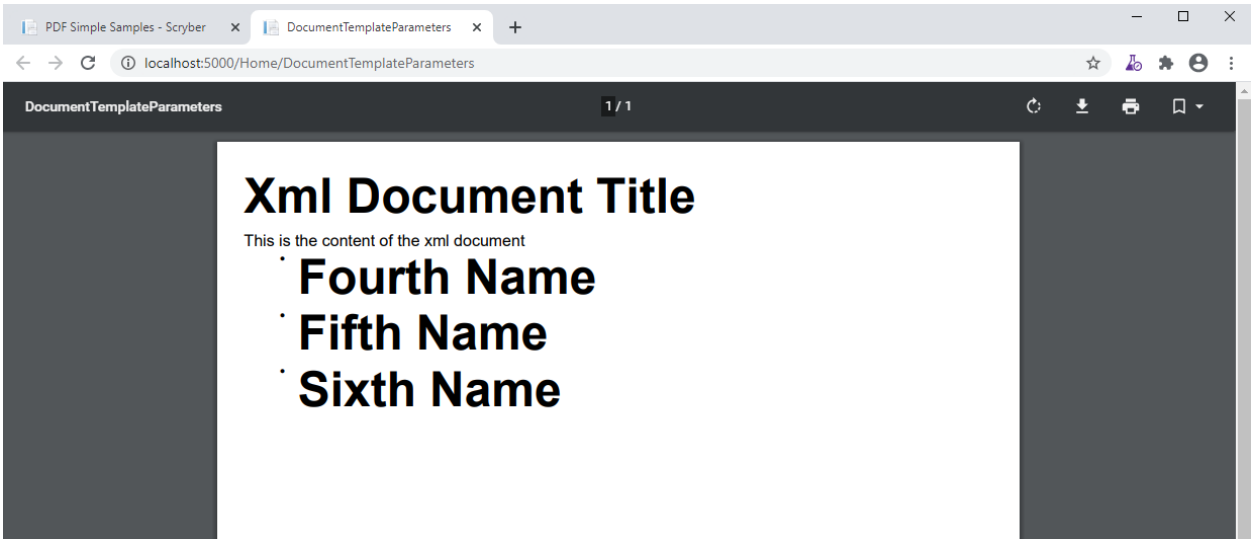
```
<?xml version="1.0" encoding="utf-8" ?>
<pdf:Document xmlns:pdf="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.
↪Components.xsd"
              xmlns:styles="http://www.scryber.co.uk/schemas/core/release/v1/
↪Scryber.Styles.xsd"
          xmlns:data="http://www.scryber.co.uk/schemas/core/release/v1/Scryber.Data.
↪xsd">
<Params>
    <!-- Declare the parameters -->
    <pdf:String-Param id="MyTitle" value="Document Title" />

    <!-- This is the xml content that will be used by default -->
    <pdf:Xml-Param id="MyData" >
    <Root>
        <Entry id="First">First Name</Entry>
        <Entry id="Second">Second Name</Entry>
        <Entry id="Third">Third Name</Entry>
    </Root>
    </pdf:Xml-Param>

    <!-- this is the template content. -->
    <pdf:Template-Param id="MyContent" >
        <pdf:Li><pdf:Text value="{xpath:text()}" /></pdf:Li>
    </pdf:Template-Param>

</Params>

<Pages>
    <!-- Use the 'MyTitle' parameter for the outline. -->
    <pdf:Page outline-title="{@:MyTitle}" styles:margins="20pt" styles:font-size="12pt
↪">
    <Content>
        <!-- And use it as the text on the heading with a visble flag and background -
↪->
        <pdf:H1  text="{@:MyTitle}" > </pdf:H1>
        <pdf:Para >This is the content of the xml document</pdf:Para>

        <pdf:Ul>
            <!-- Now we specify the template content from the parameter -->
            <data:ForEach value="{@:MyData}" select="//Root/Entry" template="
↪{@:MyContent}" ></data:ForEach>
        </pdf:Ul>


    </Content>
    </pdf:Page>
</Pages>

</pdf:Document>
```

Creating this document at runtime pulls the template data from the parameter *MyContent*

We can then change the value in code to use a different template as well as the xml (including any binding statements).

```
 [HttpGet]
public IActionResult DocumentTemplateParameters()
{
    var path = _rootPath;
    path = System.IO.Path.Combine(path, "Views", "PDF", "DocumentTemplateParameters.
↪pdfx");
    var doc = PDFDocument.ParseDocument(path);

    doc.Params["MyTitle"] = "Xml Document Title";
    var ele = new XElement("Root",
        new XElement("Entry", new XAttribute("id", "Fourth"), new XText("Fourth Name
↪")),
        new XElement("Entry", new XAttribute("id", "Fifth"), new XText("Fifth Name")),
        new XElement("Entry", new XAttribute("id", "Sixth"), new XText("Sixth Name"))
        );
    doc.Params["MyData"] = ele;

    //Just a simple example to change the template.
    doc.Params["MyTemplate"] = "<pdf:Li><pdf:H1 text='{xpath:text()}' /></pdf:Li>";

    return this.PDF(doc);
}
```

The document will then be generated with headings as the content of the list items, rather than just text values.



The following components support the use of the template attrobute to pull the value from a parameter.

- ForEach (see *Scryber.Data.PDFForEach*)
- PlaceHolder (see reference/pdf_placeholder)
- DataTemplateColumn (see reference/data_templatecolumn)
- Choose When (see reference/data_choose)
- Choose Otherwise (see reference/data_choose)
- If (see reference/data_if)

6.19.11 Passing parameters to References

The final capability for discussion is the use of the parameters when loading referenced files.

This is discussed in detail in the *Overriding and passing data* section of *Splitting into multiple files* and any type of data can be passed including templates and objects.

It starts to get really fun what you can do!

6.20 Data Binding in Scryber - td

Content to complete

6.21 Using controllers with your templates - td

Content to complete

6.22 Lifecycle of a document creation - td

There are a number of phases a document goes through when generating.

- Init
- Load
- Bind
- Layout
- Render
- Dispose

6.23 Dynamic loading of content - td

Content to complete

6.24 Namespaces and their Assemblies - td

Content coming soon

Content coming soon

6.25 Extending Scryber - td

Content coming soon

6.26 GDI Plus on dotnet core for Mac

If you are running scryber, or any other application that requires the GDI+ library, from Visual Studio for Mac or Linux, then you might encounter the following error...

```
The type initializer for 'System.Drawing.GDIPlus' threw an exception. -- ->
System.DllNotFoundException: Unable to load DLL 'gdiplus':
The specified module or one of its dependencies could not be found.
```

Unfortunately the libgdiplus libraries are not, by default, part of the DotNet Core app. (As of version 3.1.5, and 5.0 looks like it suffers the same).

6.26.1 Getting the library

Luckily there is a HomeBrew package that contains the library. (*If you don't have home brew, then the link will take you to the home page to install home brew.*)

To install the binaries use

Installing mon-libgdiplus

```
brew install mono-libgdiplus
```

This will add the libraries to the Cellar and add a link to `/usr/local`

You will then need to add link to this library to your Microsoft.NetCore.App application version you are using.

```
sudo ln -s /usr/local/lib/libgdiplus.dylib /usr/local/share/dotnet/shared/Microsoft.
↪NETCore.App/3.1.5
```

Where 3.1.5 is the latest version we were using at the time of writing. If the dotnet core app is upgraded, then it will need to be re-linked

6.26.2 Success

If you now try to rebuild and run your application or site, then this library should be found and everything work as expected.

Any problems, then give us a shout

6.27 Version History

The following change log is for developers upgrading from previous versions, or looking for new features

6.27.1 Version 1.1 Core Change log

Updated the schemas to match the new document structure Changing Document/Items to Document/Params (including in the code)

6.27.2 Version 1.0 Core Change log

This is 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

6.28 Component Reference

6.28.1 Scryber.Components.PDFDocument

The top level document has a number of capabilities

- Render Options
- Styles
- Params
- Pages

6.28.2 Scryber.Components.PDFPage

The page component is a single page. It does not normally flow onto multiple output pages.

A page can have a header and a footer, and the main content. It does not have the continuation headers or footers.

6.28.3 Scryber.Components.PDFDiv

The div component is a single block that breaks the content It normally can flow onto multiple output pages.

6.28.4 Scryber.Components.PDFList

The page component is a single page. It does not normally flow onto multiple output pages.

A page can have a header and a footer, and the main content. It does not have the continuation headers or footers.

6.28.5 Scryber.Components.PDFTable

The page component is a single page. It does not normally flow onto multiple output pages.

A page can have a header and a footer, and the main content. It does not have the continuation headers or footers.

6.28.6 Scryber.Data.PDFForEach

Content to come

6.28.7 Scryber.Data.PDFWith

Content to come

6.28.8 Scryber.Data.PDFDataGrid

Content to come

6.28.9 Scryber.Components.PDFDataList

Content to come

6.28.10 Scryber.Data.XmlDataSource

Content to come

6.28.11 Scryber.Data.ObjectCommand

Content to come

6.28.12 Scryber.Data.SqlCommand

Content to come