VectSharp.Markdown

VectSharp.Markdown can be used to render Markdown/Commonmark documents to a VectSharp `Document` object. Markdown documents are parsed using Markdig and rendered using VectSharp.

VectSharp.Markdown is released under a GPLv3 licence. The Markdig library it depends on is released under a BSD 2-Clause licence. The Highlight library used for syntax highlighting is released under a MIT licence.

Getting started

To use this library, first of all you should install the NuGet package. If you wish to enable support for embedding raster images in the document (e.g. PNG or JPEG pictures), you should also install VectSharp.MuPDFUtils.

To render a markdown source string, you can then use:

```
using System.Collections.Generic;
using VectSharp;
using VectSharp.Markdown;
using VectSharp.PDF;
//...
    string markdownSource = @"# Hello Markdown heading 1
## Hello Markdown heading 2
### Hello Markdown heading 3

Regular paragraph

    Code block

>   Block quote

1. Tight list item 1
2. Tight list item 2

* Loose bullet point 1

* Loose bullet point 2

Text can be *italic*, **bold**, or ***both***. ~~Strikethrough~~,
 su^perscripts^ and su~bscripts~ are also supported.
You can even have ++inserted++ or ==marked== text.";

    MarkdownRenderer renderer = new MarkdownRenderer();
    Document doc = renderer.Render(markdownSource, out Dictionary<string,
string> linkDestinations);
    doc.SaveAsPDF("Markdown.pdf", linkDestinations: linkDestinations);
```

In a slightly more meta example, the following code creates a PDF copy of this document:

```
using System;
using System.Collections.Generic;
using VectSharp;
using VectSharp.Markdown;
using VectSharp.PDF;
//...
    // Enables support for raster images in embedded SVG files
    VectSharp.SVG.Parser.ParseImageURI = VectSharp.MuPDFUtils.ImageURIParser.
Parser(VectSharp.SVG.Parser.ParseSVGURI);

    using (System.Net.WebClient client = new System.Net.WebClient())
    {
        string markdownSource = client.DownloadString(
"https://raw.githubusercontent.com/arklumpus/VectSharp/master/VectSharp.Markdown
/Readme.md");

        MarkdownRenderer renderer = new MarkdownRenderer()
        {
            // This uri will be used to resolve image addresses: we need them
 to point at the raw files
            BaseImageUri =
"https://raw.githubusercontent.com/arklumpus/VectSharp/master/VectSharp.Markdown
/",

            // This uri will be used to resolve link addresses: in this case,
 we don’t want to point to the raw files, but to the GitHub preview
            BaseLinkUri = new Uri(
"https://github.com/arklumpus/VectSharp/blob/master/VectSharp.Markdown/"),

            // Adds support for directly embedded raster images
            RasterImageLoader = imageFile => new VectSharp.MuPDFUtils.
RasterImageFile(imageFile)
        };

        Document doc = renderer.Render(markdownSource, out Dictionary<string,
string> linkDestinations);
        doc.SaveAsPDF("VectSharp.Markdown.pdf", linkDestinations:
 linkDestinations);
    }
```

The resulting PDF file can be accessed here.

The `MarkdownRenderer` class has many properties that can be used to tweak the rendering results (e.g. by changing the font size or font family, colours etc). These are described in the documentation.

Note the two different base uris used for images and links: since we are embedding the images, we need relative links to image files to point to the raw image file data. Thanks to the different base uri for links, instead, the GitHub preview is shown e.g. for a link to the VectSharp `Readme.md` document in the parent folder.

The first statement after the `using` directives is necessary to enable support for raster images embedded within SVG images. Instead, setting the `RasterImageLoader` property of the `MarkdownRenderer` is necessary for rendering raster images that have been directly embedded in the Markdown document. If you remove it, the VectSharp logo below will disappear.

The logo below, however, will not be affected, because it is an SVG image.

Supported Markdown features

VectSharp.Markdown uses Markdig to parse the Markdown source code. It supports most of the features of vanilla Markdown, with the notable exception of things involving raw HTML tags. Some HTML tags are supported, and are described below.

In addition to the basic features, VectSharp.Markdown supports the following extensions for Markdig (if any of these features are not rendered correctly in GitHub, look at the PDF rendering of this document to see how they should appear):

- Tables

  Both pipe tables and grid tables are supported. However:

  - Tables always take up all the available horizontal space of in the page.
  - Table cells are always left-aligned.
  - The vertical alignment of table cells can be specified on a per-document basis, using the `TableVAlign` property of the `MarkdownRenderer`.
  - Cells spanning multiple rows are not supported (cells spanning multiple columns are fine).

- Extra emphasis

  - `~~` strikethrough (~~example~~)
  - `~` subscript (exa$_{mple}$)
  - `^` superscript (exa^{mple})
  - `++` inserted (example)
  - `==` marked (example)

- Special attributes

  Only the `id` attribute is supported, which can then be used to reference the element in a link. E.g.

  ```
  # Heading {#target}

  This is a [link](#target) to the heading above.
  ```

- Auto identifiers

  This makes it possible to refer to headings without having to assign an id to each one. E.g.:

  ```
  # This is a heading

  This is a [link](#this-is-a-heading) to the heading above.
  ```

- Auto links
- Task lists

  These are non-interactive. E.g.:

  ```
  * [ ] Unchecked item
  ```

```
* [X] Checked item
* [ ] Unchecked item 2
```

- [ ] Unchecked item
- [x] Checked item
- [ ] Unchecked item 2

- [Extra bullet lists]()
- [Citations]()

  A citation is delimited by `""`, e.g. `""example""` becomes *example*. Note that figures and footers are not supported.

- [Mathematics]()

  Both inlines and block maths are supported. These are rendered by sending a query to `https://render.githubusercontent.com/render/math` with the maths text.

  For example, `$x^2$` results in a query to `https://render.githubusercontent.com/render/math?math=x^2` and is rendered as x^2. If you wish to render maths differently, you can set the `ImageUriResolver` property of the `MarkdownRenderer` to a custom method and intercept these queries.

- [SmartyPants]()

  This replaces quotes with the Unicode character corresponding to the appropriate smart quotes, as well as dashes and ellipses with the appropriate Unicode characters.

  E.g.: “double quotes”, ‘single quotes’, medium dash – long dash — ellipsis ...

- Syntax highlighting

  Syntax highlighting in VectSharp.Markdown uses the [Highlight]() library. The supported languages are:

  - ASPX
  - C
  - C++
  - C#
  - COBOL
  - Eiffel
  - Fortran
  - Haskell
  - HTML
  - Java
  - JavaScript
  - Mercury
  - MSIL
  - Pascal
  - Perl
  - PHP
  - Python
  - Ruby
  - SQL
  - Visual Basic
  - VBScript
  - VB.NET
  - XML

  Custom syntax highlighting can be achieved by changing the value of the `SyntaxHighlighter` property of the `MarkdownRenderer`. This is a method that takes as arguments the source code to highlight and the language, and outputs a list of lists of `FormattedString` objects.

  Each `FormattedString` object represents a run of characters with the same style (i.e. colour, font weight and font style); each list of `FormattedString`s represents a single line of code, and the list of lists represents the whole code block.

By replacing the value of this property with a different method, it is possible to add custom syntax highlighting for other languages. To disable syntax highlighting, set this property to a method always returning `null`.

When a string containing Markdown code is passed to VectSharp.Markdown, by default the code is parsed with all these extensions enabled. If you wish to disable them (or to enable only a subset of them), you can use the overloads that take a `MarkdownDocument` and parse the document yourself:

```
using System.Collections.Generic;
using VectSharp;
using VectSharp.Markdown;
using VectSharp.PDF;
//...
    string markdownSource = ...

    MarkdownDocument markdownDocument = Markdig.Markdown.Parse(markdownSource,
new MarkdownPipelineBuilder().UseSmartyPants().Build());

    MarkdownRenderer renderer = new MarkdownRenderer();
    Document doc = renderer.Render(markdownDocument, out Dictionary<string,
string> linkDestinations);
    doc.SaveAsPDF("Markdown.pdf", linkDestinations: linkDestinations);
```

Supported HTML tags

- `<img>` or `<image>` tags for including images are supported. The tag must contain an `src` attribute specifying the image file, and can contain an `align` attribute with value `left`, `center` or `right`, specifying the alignment of the image in the page.

  If no `align` attribute is provided, the image is treated as being inline (except if it is the only element in its block). Otherwise:

  - If `align` is `center`, the centered image interrupts the flow of the text.
  - If `align` is `left` or `right`, the image is rendered as a floating element on the respective side. Text is arranged in such a way that it flows around the image.

- `<p>` tags are supported only to provide alignment to `<img>` tags. For example `<p align="center"><img src="image.png"/></p>` is equivalent to `<img align="center" src="image.png"/>`. **Any text included within `<p></p>` tags is ignored** (as is text within any HTML block).

- `<a>` tags are supported only to provide anchors for regular Markdown links (e.g. `<a name="linkTarget"></a>`). Things like `<a href="https://www.github.com/"></a>` **will not work**!

- `<br>` tags can be used for line breaks and page breaks. A simple `<br/>` tag signifies a line break, while `<br type="page"/>` can be used to force a page break.

Single-page rendering

In addition to the `Render` method, which creates a `Document` with multiple pages of the specified size (by default, corresponding to an A4 sheet of paper), the `MarkdownRenderer` class also includes the `RenderSinglePage` method. This method renders the document to a single page with the specified width. The final height of the page is determined so that it contains the entire document (with the specified `Margins`):

```
using System.Collections.Generic;
using VectSharp;
using VectSharp.Markdown;
using VectSharp.SVG;
//...
    string markdownSource = ...

    MarkdownRenderer renderer = new MarkdownRenderer();
    Page pag = renderer.RenderSinglePage(markdownSource, 595, out Dictionary<
string, string> linkDestinations);
    pag.SaveAsSVG("Markdown.svg", linkDestinations: linkDestinations);
```

Tests

The `MarkdownExamples` project in this repository uses VectSharp.Markdown to render the test cases from the CommonMark spec. It produces 33 HTML files, containing 20 tests each. Each test is represented by a table with the raw Markdown source on the left, the reference HTML in the middle, and an SVG rendering produced by VectSharp.Markdown on the right. If everything is working correctly, the SVG images should look very similar to the reference HTML.