VectSharp: a light library for C# vector graphics

License LGPL v3 nuget v2.5.0

Introduction

VectSharp is a library to create vector graphics (including text) in C#, without too many dependencies.

VectSharp is written using .NET Core, and is available for Mac, Windows and Linux. Since version 2.0.0, it is released under an LGPLv3 license. It includes 14 standard fonts, originally released under an ASL-2.0 license.

It includes an abstract layer on top of which output layers can be written. Currently, there are five available output layers:

- **VectSharp.PDF** produces PDF documents.
- **VectSharp.Canvas** produces an `Avalonia.Controls.Canvas` object (https://avaloniaui.net/docs/controls/canvas) containing the rendered graphics objects.
- **VectSharp.SVG** produces vector graphics in SVG format.
- **VectSharp.Raster** produces raster images in PNG format, (this is done by rendering the image to a PDF document, and then using the MuPDFCore library to render the PDF). Since version 2.0.0, VectSharp.Raster is released under an AGPLv3 licence.
- **VectSharp.Raster.ImageSharp** produces raster images in multiple formats (BMP, GIF, JPEG, PBM, PNG, TGA, TIFF, WebP) using the SixLabors.ImageSharp library.

VectSharp.Raster and VectSharp.Raster.ImageSharp are somewhat overlapping, as both of them can be used to create PNG images. However, VectSharp.Raster is much faster, though it only supports the PNG format. Instead, VectSharp.Raster.ImageSharp is slower, but supports more formats and has a more permissive licence. Another difference is that VectSharp.Raster carries a native dependency (through MuPDFCore), while VectSharp.ImageSharp does not.

Furthermore:

- The **VectSharp.Plots** package contains classes and methods to draw plots (such as scatter plots, line charts, bar charts, box plots, function plots, and more).

- **VectSharp.ThreeD** adds support for 3D vector and raster graphics.

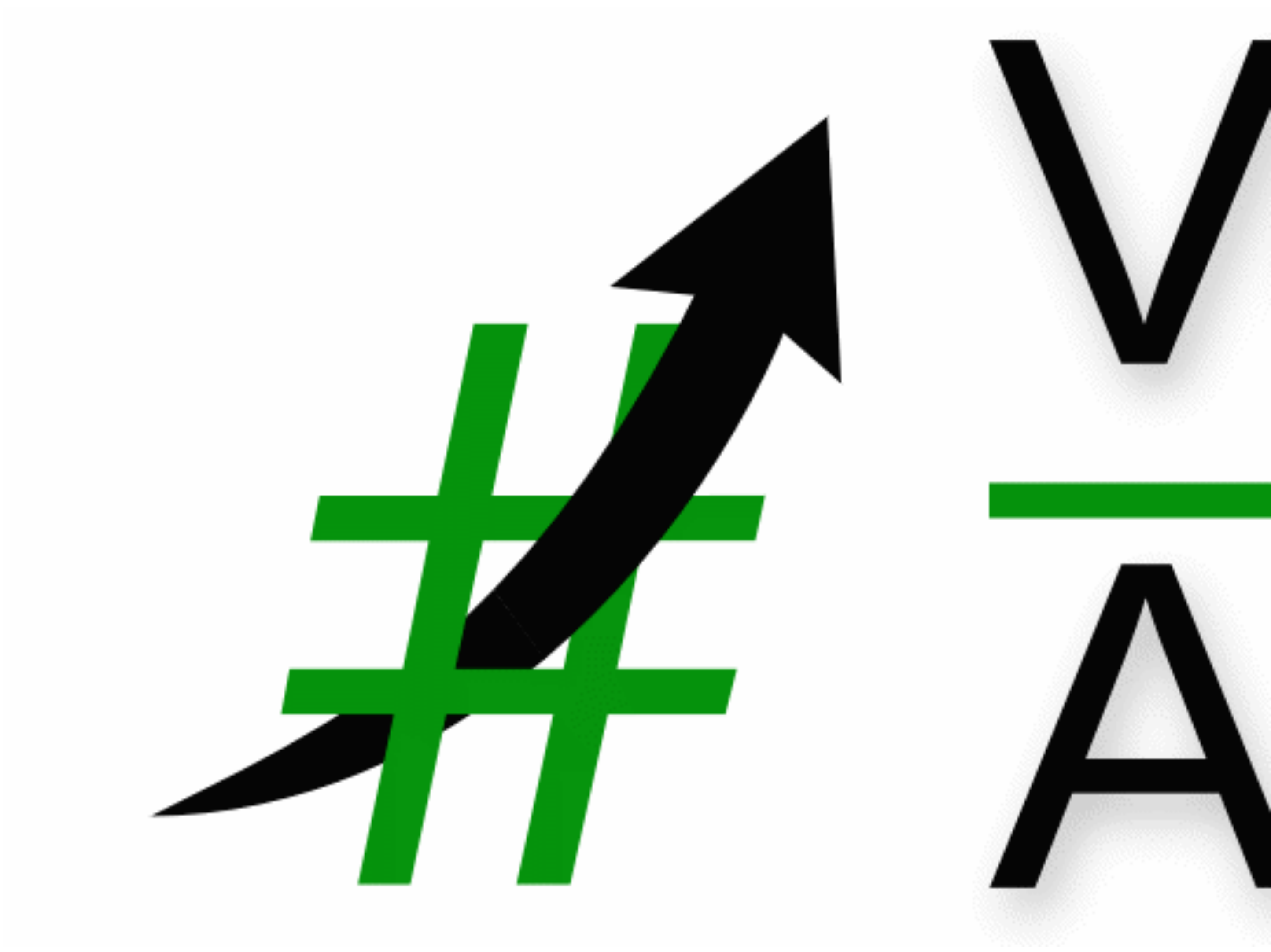
- **VectSharp.Markdown** can be used to transform Markdown documents into VectSharp objects, that can then be exported e.g. as PDF or SVG files, or displayed in an Avalonia `Canvas`. **VectSharp.MarkdownCanvas** uses VectSharp.Markdown to render Markdown documents in Avalonia applications (an example of this is in the MarkdownViewerDemo project).

- **VectSharp.MuPDFUtils**, released under an AGPLv3 licence, contains some utility functions that use MuPDFCore to make it possible to include in VectSharp graphics images in various formats.

- **VectSharp.ImageSharpUtils** adds the same capabilities as VectSharp.MuPDFUtils, using ImageSharp instead of MuPDFCore; as a result, it is released under a more permissive LGPLv3 licence.

- **VectSharp.Fonts.Nimbus** is a package released under a GPLv3 license, which contains the standard fonts that were used in VectSharp before version 2.0.0. Since these fonts are released under a GPL license, they had to be replaced when the VectSharp license changed to LGPL. See the Font libraries below for information on how to re-enable these fonts.

- The **Animation** class (provided in the base VectSharp package) can be used to create animations that can be saved as animated GIFs (using VectSharp.Raster.ImageSharp), SVGs (using VectSharp.SVG) and PNGs (using VectSharp.Raster or VectSharp.Raster.ImageSharp).



Installing VectSharp

To include VectSharp in your project, you will need one of the output layer NuGet packages: VectSharp.PDF, VectSharp.Canvas, VectSharp.Raster, VectSharp.Raster.ImageSharp, or VectSharp.SVG. You will need VectSharp.ThreeD to work with 3D graphics or VectSharp.Plots to create plots. You may want the VectSharp.MuPDFUtils package if you wish to manipulate raster images, and the VectSharp.Fonts.Nimbus if you want to restore the GPL-licensed fonts used in previous versions of the library.

Usage

You can find detailed documentation for the VectSharp library, **including interactive examples**, at the documentation website. A comprehensive API reference is also available, both as a website and as a PDF manual.

In general, working with VectSharp involves: creating a `Document`, adding `Page`s, drawing to the `Page`s' `Graphics` objects and, finally, exporting them to a PDF document, `Canvas`, PNG image or SVG document.

- Create a `Document`:

```
using VectSharp;
// ...
```

```csharp
Document doc = new Document();
```

- Add a `Page`:

```csharp
doc.Pages.Add(new Page(1000, 1000));
```

- Draw to the `Page`'s `Graphics` object:

```csharp
Graphics gpr = doc.Pages.Last().Graphics;
gpr.FillRectangle(100, 100, 800, 800, Colour.FromRgb(128, 128, 128));
```

- Save as PDF document:

```csharp
using VectSharp.PDF;
//...
doc.SaveAsPDF(@"Sample.pdf");
```

- Export the graphics to a `Canvas`:

```csharp
using VectSharp.Canvas;
//...
Avalonia.Controls.Canvas can = doc.Pages.Last().PaintToCanvas();
```

- Export the graphics to a `Canvas`, using a multi-layer, multi-threaded, triple-buffered renderer based on SkiaSharp (which provides the best performance if you wish e.g. to place the canvas within a `ZoomBorder`):

```csharp
using VectSharp.Canvas;
//...
// A single page
Avalonia.Controls.Canvas can = doc.Pages.Last().PaintToSKCanvas();

// The whole document - each page will correspond to a layer
Avalonia.Controls.Canvas can = doc.PaintToSKCanvas();
```

- Save as a PNG image:

```csharp
using VectSharp.Raster;
//...
doc.Pages.Last().SaveAsPNG(@"Sample.png");
```

- Save as a JPEG image:

```csharp
using VectSharp.Raster.ImageSharp;
//...
doc.Pages.Last().SaveAsImage(@"Sample.jpg");
```

- Save as an SVG document:

```csharp
using VectSharp.SVG;
//...
doc.Pages.Last().SaveAsSVG(@"Sample.svg");
```

- PDF and SVG documents support both internal and external links:

```csharp
using VectSharp;
using VectSharp.PDF;
using VectSharp.SVG;
//...
Document document = new Document();
Page page = new Page(1000, 1000);
document.Pages.Add(page);

```

```
page.Graphics.FillRectangle(100, 100, 800, 50, Colour.FromRgb(128, 128, 128),
 tag: "linkToGitHub");
page.Graphics.FillRectangle(100, 300, 800, 50, Colour.FromRgb(255, 0, 0), tag:
"linkToBlueRectangle");
page.Graphics.FillRectangle(100, 850, 800, 50, Colour.FromRgb(0, 0, 255), tag:
"blueRectangle");

Dictionary<string, string> links = new Dictionary<string, string>() { {
"linkToGitHub", "https://github.com/" }, { "linkToBlueRectangle",
"#blueRectangle" } };

page.SaveAsSVG(@"Links.svg", linkDestinations: links);
document.SaveAsPDF(@"Links.pdf", linkDestinations: links);
```

This code produces a document with three rectangles: the grey one at the top links to the GitHub home page, while the red one in the middle is a hyperlink to the blue one at the bottom. Links in PDF documents can refer to objects that are in a different page than the one containing the link.

The public classes and methods are fully documented (with interactive examples created using Blazor), and you can find a (much) more detailed code example in MainWindow.xaml.cs. A detailed guide about 3D graphics in VectSharp.ThreeD is available in the `VectSharp.ThreeD` folder. Further example code for animations is available in the DemoAnimation project.

Font libraries

Since version 2.0.0, font names are resolved using a "font library". This is a class that implements the `VectSharp.IFontLibrary` interface, providing methods to obtain a `FontFamily` object from a `string` or a `FontFamily.StandardFontFamilies` enumeration. The default font library included in VectSharp uses the embedded fonts (Arimo, Tinos, Cousine) as the standard font families.

In practice, assuming you want to use the default font library, you have the following options to create a `FontFamily` object:

```
using VectSharp;

// ...

FontFamily helvetica = FontFamily.ResolveFontFamily(
FontFamily.StandardFontFamilies.Helvetica); // Will resolve to the Arimo font
 family.
FontFamily times = FontFamily.ResolveFontFamily("Times-Roman"); // Will resolve
 to the Tinos font family.
```

These replace the `FontFamily(string)` and `FontFamily(StandardFontFamilies)` constructors of previous versions of VectSharp. Overloads of this method let you specify a list of "fallback" fonts that will be used if the first font you specify is not available.

If you wish, you can replace the default font library with a different one; this will change the way font families are resolved. For example, after installing the VectSharp.Fonts.Nimbus NuGet package, you can do:

```
using VectSharp;

// ...

FontFamily.DefaultFontLibrary = VectSharp.Fonts.Nimbus.Library;

FontFamily helvetica = FontFamily.ResolveFontFamily(
FontFamily.StandardFontFamilies.Helvetica); // Will resolve to the Nimbus Sans
```

```
 L font family.
FontFamily times = FontFamily.ResolveFontFamily("Times-Roman"); // Will resolve
 to the Nimbus Roman No 9 L font family.
```

This will let you re-enable the fonts that were used in previous versions of VectSharp.

You can also use multiple font libraries in the same project. Again, assuming you have installed the VectSharp.Fonts.Nimbus NuGet package:

```
using VectSharp;

FontFamily helvetica1 = FontFamily.ResolveFontFamily(
FontFamily.StandardFontFamilies.Helvetica); // Will resolve to the Arimo font
 family.
FontFamily times1 = FontFamily.ResolveFontFamily("Times-Roman"); // Will
 resolve to the Tinos font family.

FontFamily helvetica2 = VectSharp.Fonts.Nimbus.ResolveFontFamily(
FontFamily.StandardFontFamilies.Helvetica); // Will resolve to the Nimbus Sans
 L font family.
FontFamily times2 = VectSharp.Fonts.Nimbus.ResolveFontFamily("Times-Roman"); //
 Will resolve to the Nimbus Roman No 9 L font family.
```

Finally, you can create your own font library class (which could implement things such as dowloading fonts from Google Fonts, or finding them in the user’s system font directory…) by creating a class that implements the `IFontLibrary` interface or that extends the `FontLibrary` class (in this latter case, you get a default implementation for the `ResolveFontFamily` overloads that use a list of fallback fonts).

Creating new output layers

VectSharp can be easily extended to provide additional output layers. To do so:

1. Create a new class implementing the `IGraphicsContext` interface.
2. Provide an extension method to either the `Page` or `Document` types.
3. Somewhere in the extension method, call the `CopyToIGraphicsContext` method on the `Graphics` object of the `Page` s.
4. Opportunely save or return the rendered result.

Compiling VectSharp from source

The VectSharp source code includes an example project (*VectSharp.Demo*) presenting how VectSharp can be used to produce graphics.

To be able to compile VectSharp from source, you will need to install the latest .NET SDK for your operating system.

You can use Microsoft Visual Studio to compile the program. The following instructions will cover compiling VectSharp from the command line, instead.

First of all, you will need to download the VectSharp source code: VectSharp.tar.gz and extract it somewhere.

Windows

Open a command-line window in the folder where you have extracted the source code, and type:

```
BuildDemo <Target>
```

Where `<Target>` can be one of `Win-x64` , `Linux-x64` or `Mac-x64` depending on which platform you wish to generate executables for.

In the Release folder and in the appropriate subfolder for the target platform you selected, you will find the compiled program.

macOS and Linux

Open a terminal in the folder where you have extracted the source code, and type:

```
./BuildDemo.sh <Target>
```

Where `<Target>` can be one of `Win-x64`, `Linux-x64` or `Mac-x64` depending on which platform you wish to generate executables for.

In the Release folder and in the appropriate subfolder for the target platform you selected, you will find the compiled program.

If you receive an error about permissions being denied, try typing `chmod +x BuildDemo.sh` first.

Note about VectSharp.MuPDFUtils and .NET Framework

If you wish to use VectSharp.MuPDFUtils in a .NET Framework project, you will need to manually copy the native MuPDFWrapper library for the platform you are using to the executable directory (this is done automatically if you target .NET core).

One way to obtain the appropriate library files is:

1. Manually download the NuGet package for MuPFDCore (click on the “Download package” link on the right).
2. Rename the `.nupkg` file so that it has a `.zip` extension.
3. Extract the zip file.
4. Within the extracted folder, the library files are in the `runtimes/xxx-yyy/native/` folder, where `xxx` is either `linux`, `osx` or `win`, depending on the platform you are using, and `yyy` is `x64`, `x86` or `arm64` depending on the architecture.

Make sure you copy the appropriate file to the same folder as the executable!