

PDF Metamorphosis .Net

(Multi-platform .Net library)

[SautinSoft](#)

Linux development manual

Table of Contents

1. Preparing environment	2
2. Creating "Convert RTF/DOCX to PDF" app.....	4

1. Preparing environment

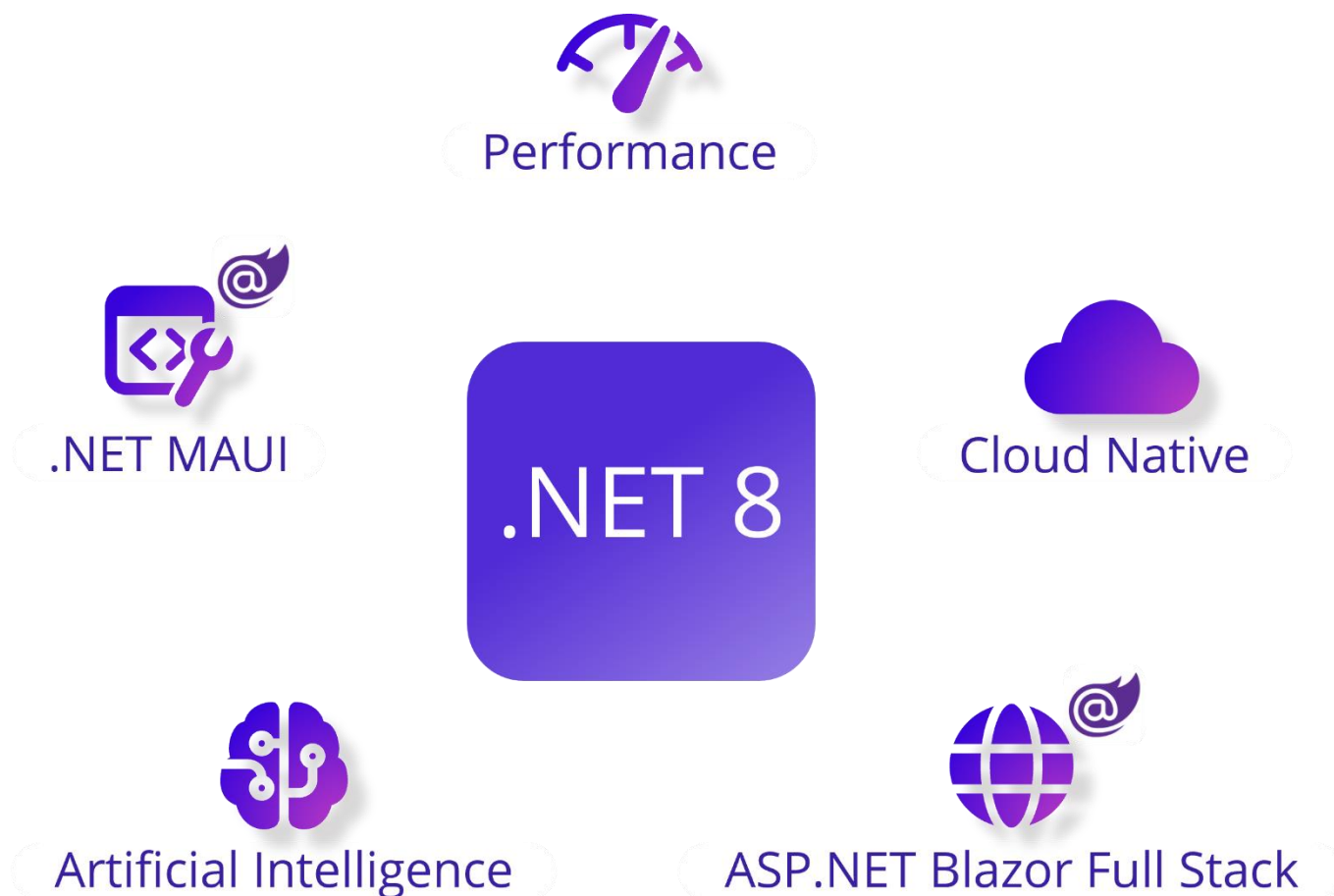
In order to build multi-platform applications using .NET Core on Linux, the first steps are for installing in our Linux machine the required tools.

We need to install .NET Core SDK from Microsoft and to allow us to develop easier, we will install an advance editor with a lot of features, Visual Studio Code from Microsoft.

Both installations are very easy and the detailed description can be found by these two links:

[Install .NET Core SDK for Linux.](#)

Please check the current version of .Net Core (2.X, 3.X)



[Install VS Code for Linux.](#)

Once installed VS Code, you need to install a C# extension to facilitate us to code and debugging:

Install [C# extension](#).

In next paragraphs we will explain in detail how to create simple console application. All of them are based on this VS Code guide:

[Get Started with C# and Visual Studio Code](#)

Not only is possible to create .NET Core applications that will run on Linux using Linux as a developing platform. It is also possible to create it using a Windows machine and any modern Visual Studio version, as Microsoft Visual Studio Community 2022.

The image shows the Visual Studio Code website on the left and a screenshot of the Visual Studio Code IDE on the right.

Visual Studio Code Website:

- Header: Visual Studio Code, Docs, Updates, Blog, API, Extensions, FAQ, Learn, Search Docs, Download.
- Message: Version 1.84 is now available! Read about the new features and fixes from October.
- Section: Code editing. Redefined.
- Text: Free. Built on open source. Runs everywhere.
- Buttons: Download for Windows (Stable Build), Web, Insiders edition, or other platforms.
- Text: By using VS Code, you agree to its license and privacy statement.

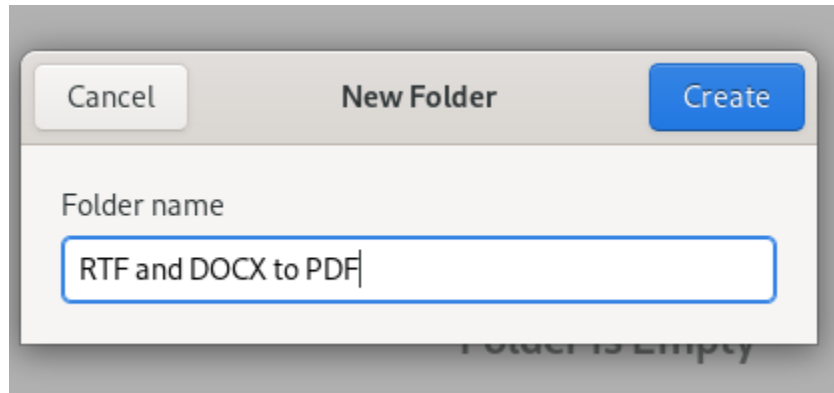
Visual Studio Code IDE Screenshot:

- File Explorer: Extensions: Marketplace, @sortinstalls.
- Search: Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support, Vetur, C#.
- Code Editor: JS serviceWorker.js, register, window.addEventListener('load') callback, checkValidServiceWorker(swUrl, config);, // Add some additional logging to localhost, p..., // service worker/PWA documentation., navigator.serviceWorker.ready.then(() => {, product, productSub, removeSiteSpecificTrackingException, removeWebWideTrackingException, requestMediaKeySystemAccess, sendBeacon, serviceWorker (property) Navigator.serviceWorke..., storage, storeSiteSpecificTrackingException, storeWebWideTrackingException, userAgent, vendor, function registerValidSW(swUrl, config) {, navigator.serviceWorker, .register(swUrl), .then(registration => {.
- Terminal: 1: node, You can now view create-react-app in the browser., Local: http://localhost:3000/, On Your Network: http://10.211.55.3:3000/, Note that the development build is not optimized.

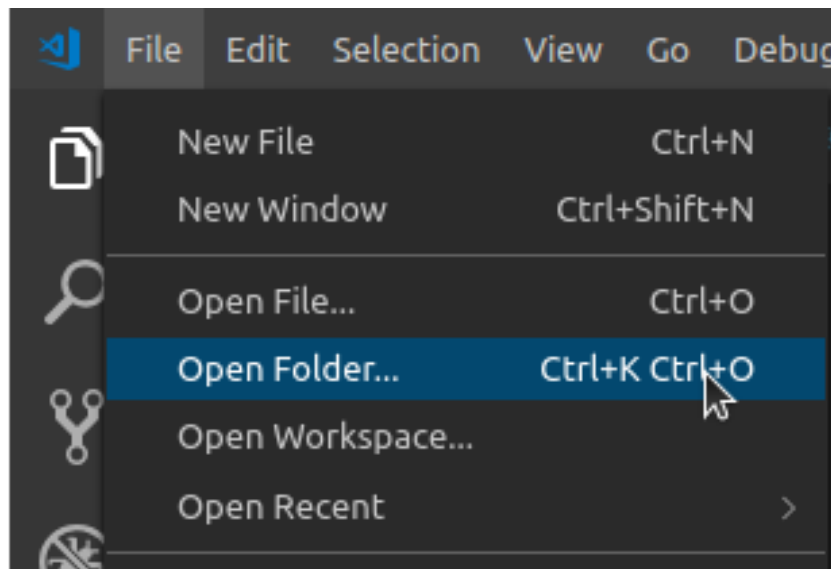
2. Creating “Convert RTF/DOCX to PDF” app

Create a new folder in your Linux machine with the name ***RTF and DOCX to PDF***.

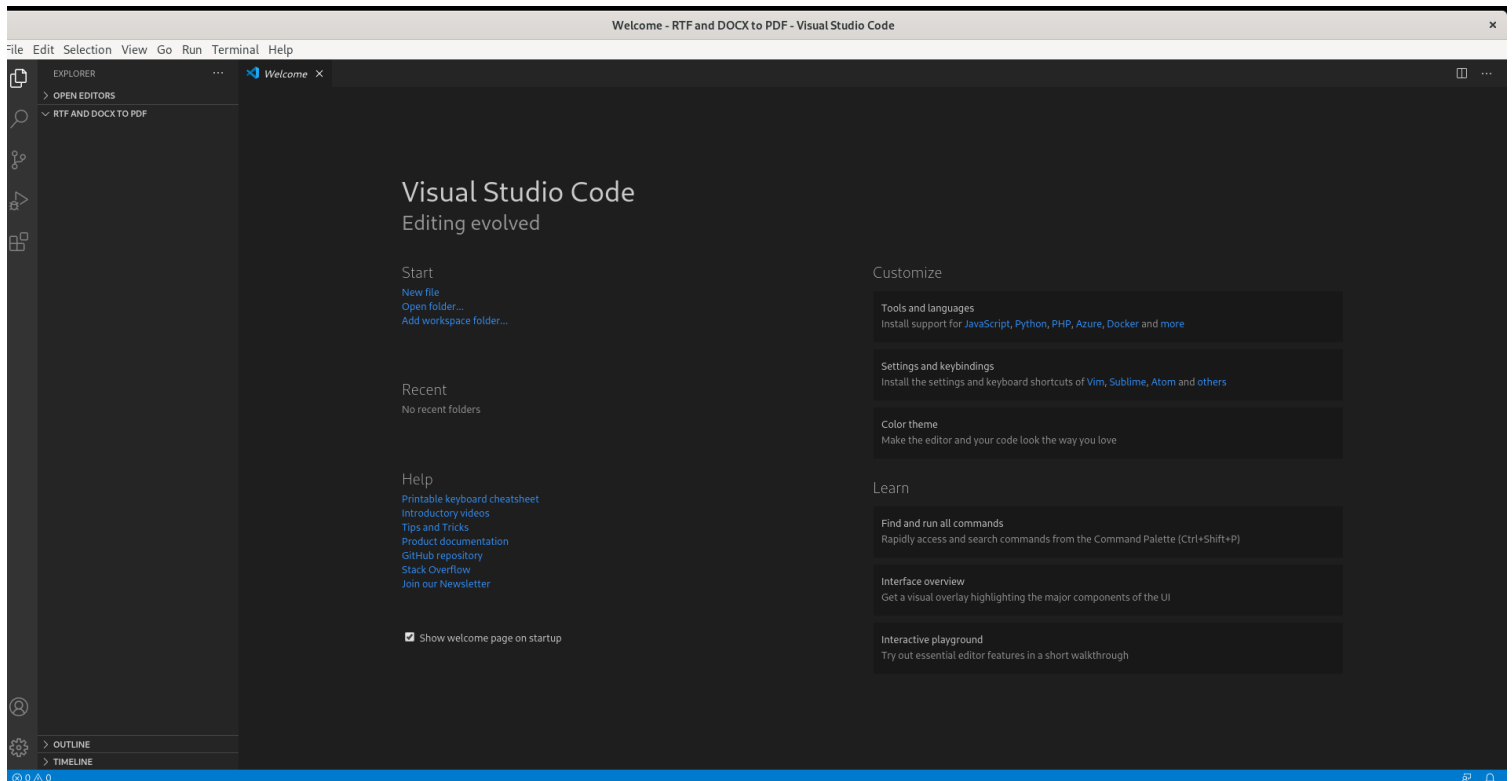
For example, let’s create the folder “***RTF and DOCX to PDF***” on the Desktop (Right click-> New Folder):



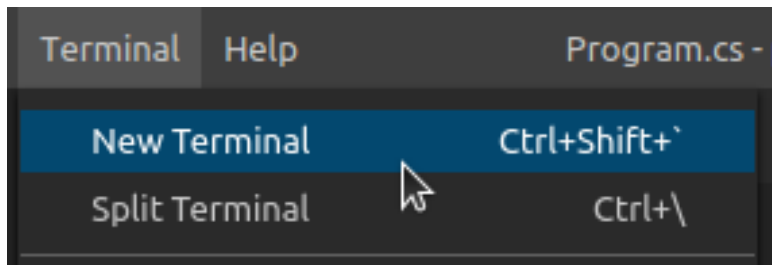
Open VS Code and click in the menu ***File->Open Folder***. From the dialog, open the folder you’ve created previously:



Next you will see the similar screen:

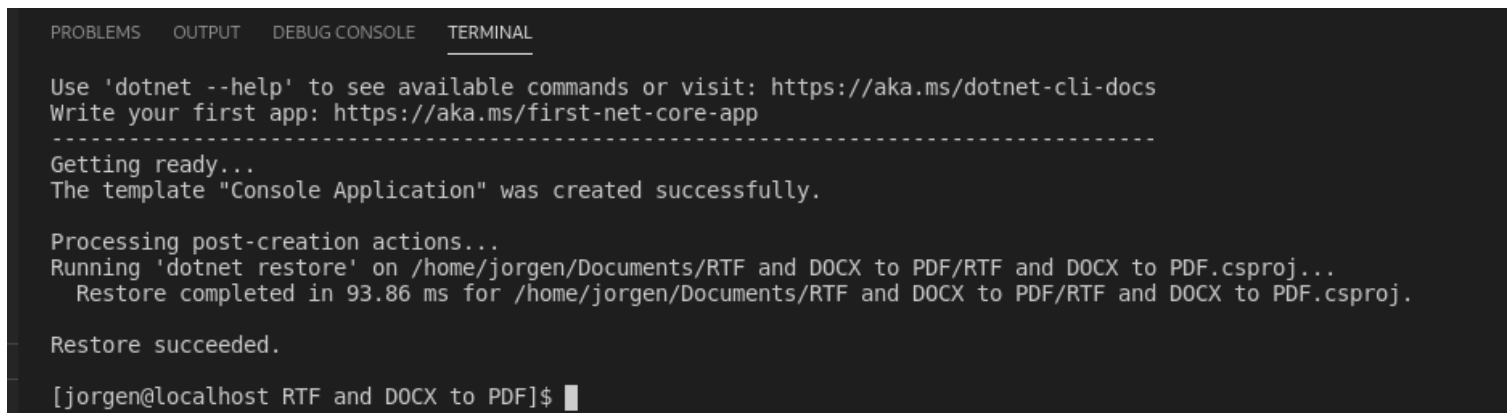


Now, open the integrated console – the Terminal: follow to the menu **Terminal** -> **New Terminal** (or press Ctrl+Shift+`):



Create a new console application, using **dotnet** command.

Type this command in the Terminal console: **dotnet new console**



A new simple ***Hello world!*** console application has been created. To execute it, type this command: ***dotnet run***

```
Restore succeeded.

[jorgen@localhost RTF and DOCX to PDF]$ dotnet run
Hello World!
[jorgen@localhost RTF and DOCX to PDF]$
```

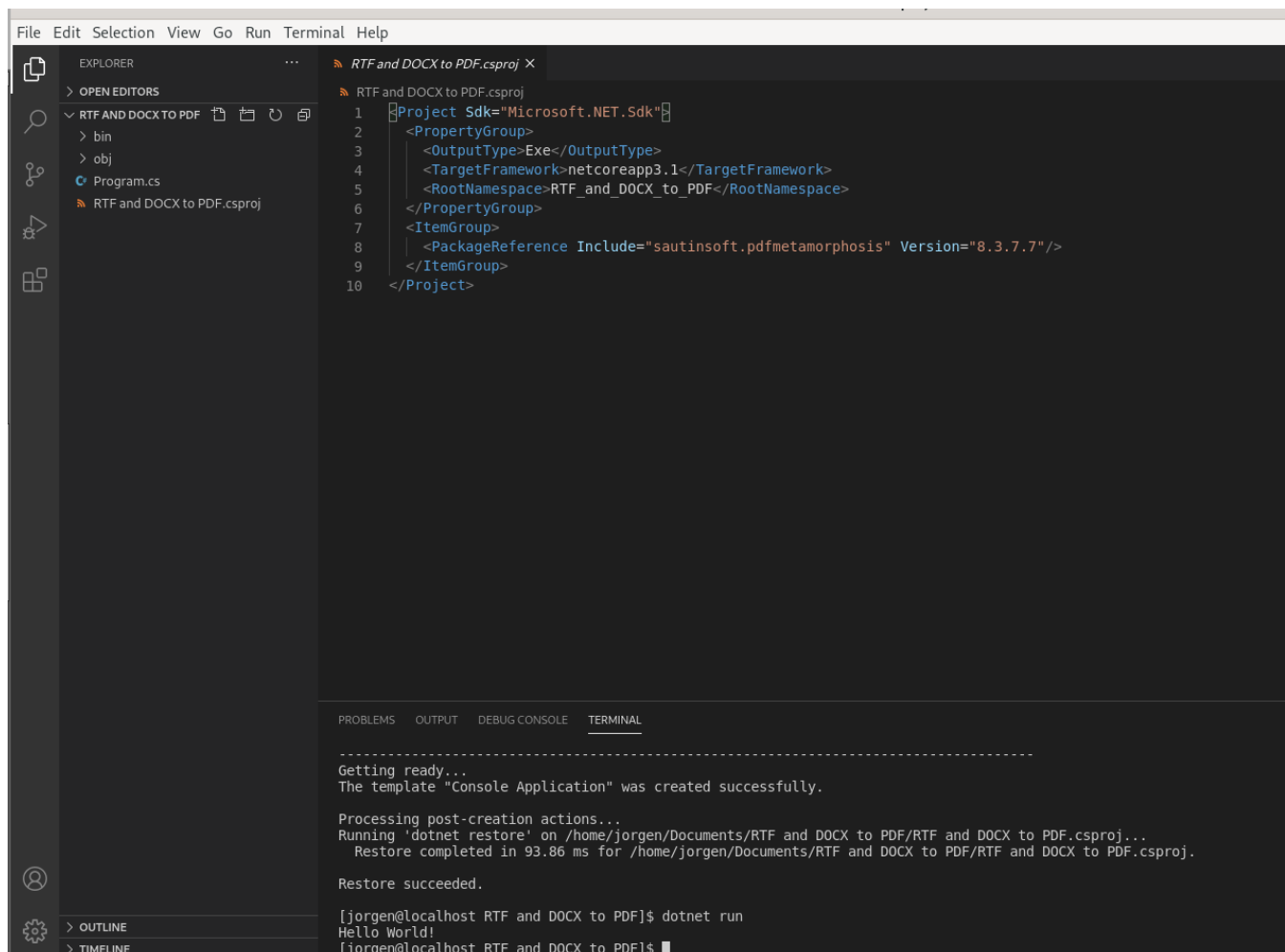
You can see the typical “Hello world!” message.

Now we are going to convert this simple application into something more interesting.

We’ll transform it into an application that will convert html, rtf, docx files to pdf files.

First of all, we need to add the package reference to the ***sautinsoft.pdfmetamorphosis*** assembly using Nuget.

In order to do it, follow to the ***Explorer*** and open project file “***RTF and DOCX to PDF.csproj***” within VS Code to edit it:



Add these lines into the **.csproj** file:

```
<ItemGroup>
  <PackageReference Include="sautinsoft.pdfmetamorphosis" Version="*" />
  <PackageReference Include="Pkcs11Interop" Version="5.1.2" />
  <PackageReference Include="Portable.BouncyCastle" Version="1.9.0" />
  <PackageReference Include="SkiaSharp" Version="2.88.6" />
  <PackageReference Include="SkiaSharp.HarfBuzz" Version="2.88.6" />
  <PackageReference Include="Svg.Skia" Version="1.0.0.3" />
  <PackageReference Include="System.IO.Packaging" Version="4.4.0" />
  <PackageReference Include="System.Text.Encoding.CodePages" Version="4.5.0" />
  <PackageReference Include="System.Xml.XPath.XmlDocument" Version="4.3.0" />
  <PackageReference Include="SkiaSharp.NativeAssets.Linux" Version="2.88.6" />
</ItemGroup>
```

It's the reference to **sautinsoft.pdfmetamorphosis** package from Nuget.

At the moment of writing this manual, the latest version of **sautinsoft.pdfmetamorphosis** was 2024.X. But you may specify the latest version, to know what is the latest, follow:

<https://www.nuget.org/packages/sautinsoft.pdfmetamorphosis>

At once as we've added the package reference, we have to save the **"RTF and DOCX to PDF.csproj"** and restore the added package.

Follow to the **Terminal** and type the command: **dotnet restore**

The template "Console Application" was created successfully.

Processing post-creation actions...

Running 'dotnet restore' on /home/jorgen/Documents/RTF and DOCX to PDF/RTF and DOCX to PDF.csproj...

Restore completed in 93.86 ms for /home/jorgen/Documents/RTF and DOCX to PDF/RTF and DOCX to PDF.csproj.

Restore succeeded.

[jorgen@localhost RTF and DOCX to PDF]\$ dotnet run

Hello World!

[jorgen@localhost RTF and DOCX to PDF]\$ dotnet restore

Restore completed in 26.28 sec for /home/jorgen/Documents/RTF and DOCX to PDF/RTF and DOCX to PDF.csproj.

[jorgen@localhost RTF and DOCX to PDF]\$ █

Important. This information is valid for versions up to 2024.X

There are a lot of Linux varieties: Suse, Fedora, Debian, Ubuntu, etc.

In addition, there are cloud platforms: AWS Lambda, Google Cloud, Azure, Apex, etc.

Because our dll works with Graphics and using GDI+ API, you need to check, that your system has [System.Drawing.Common](#) is the graphics library which ships as part of .NET Core. On macOS and Linux, it uses [libgdiplus](#) as its back-end.

There are existing ways to install libgdiplus on macOS and Linux. On macOS, you can use the [mono-libgdiplus](#) Homebrew package; on Ubuntu Linux, you can install the [libgdiplus-dev](#) package.

🔗 Using libgdiplus on Ubuntu Linux

You can install libgdiplus on Ubuntu Linux using the Quamotion PPA. Follow these steps:

```
sudo add-apt-repository ppa:quamotion/ppa
sudo apt-get update
sudo apt-get install -y libgdiplus
```

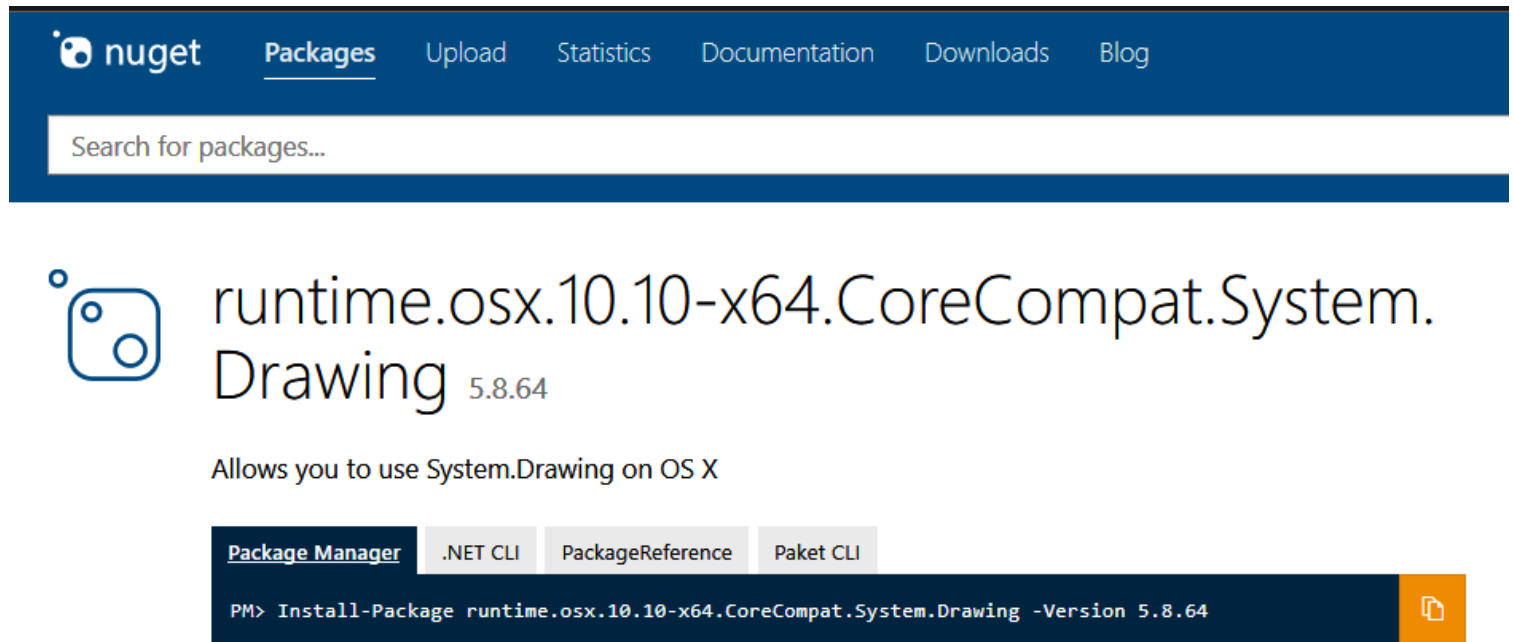
Using libgdiplus on macOS

On macOS, add a reference to the [runtime.osx.10.10-x64.CoreCompat.System.Drawing](#) package:

```
dotnet add package runtime.osx.10.10-x64.CoreCompat.System.Drawing
```

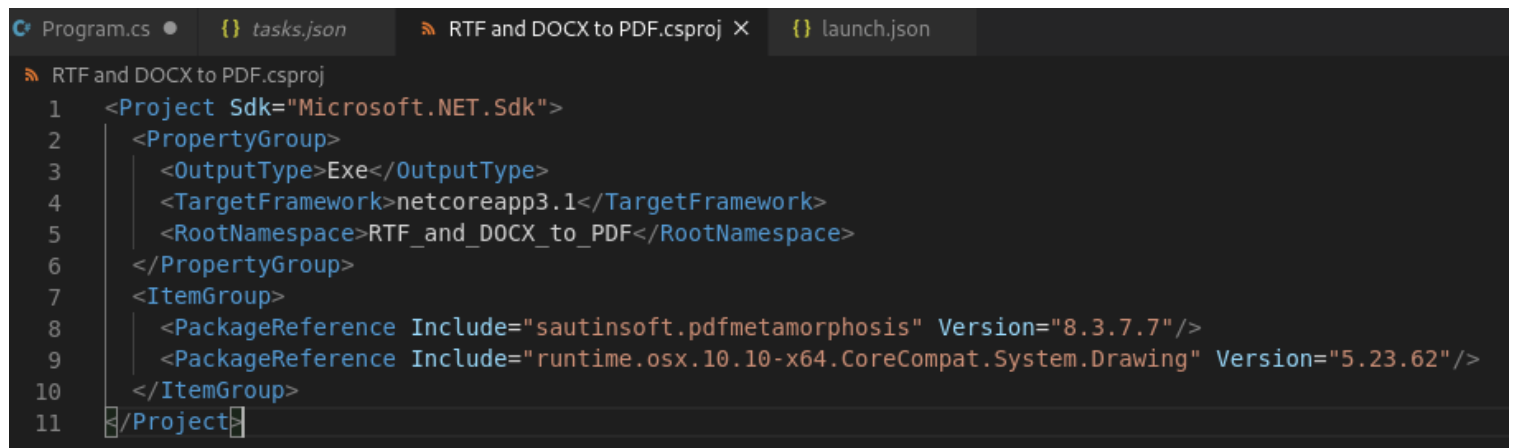
If you have installed LibGdiPlus' dll and it still doesn't work. Please add (update) this string in your project:


```
<PackageReference Include="runtime.osx.10.10-x64.CoreCompat.System.Drawing"
Version="5.23.62"/>
```



The image shows the NuGet package page for `runtime.osx.10.10-x64.CoreCompat.System.Drawing` version 5.8.64. The page has a blue header with the NuGet logo and navigation links: Packages, Upload, Statistics, Documentation, Downloads, and Blog. Below the header is a search bar. The main content area features the package icon (a blue square with two circles), the package name, and version number. A description states: "Allows you to use System.Drawing on OS X". Below this is a tabbed interface with four tabs: Package Manager (selected), .NET CLI, PackageReference, and Paket CLI. The Package Manager tab shows a command to install the package: `PM> Install-Package runtime.osx.10.10-x64.CoreCompat.System.Drawing -Version 5.8.64`. There is a copy icon to the right of the command.

As a result, your project should contain:

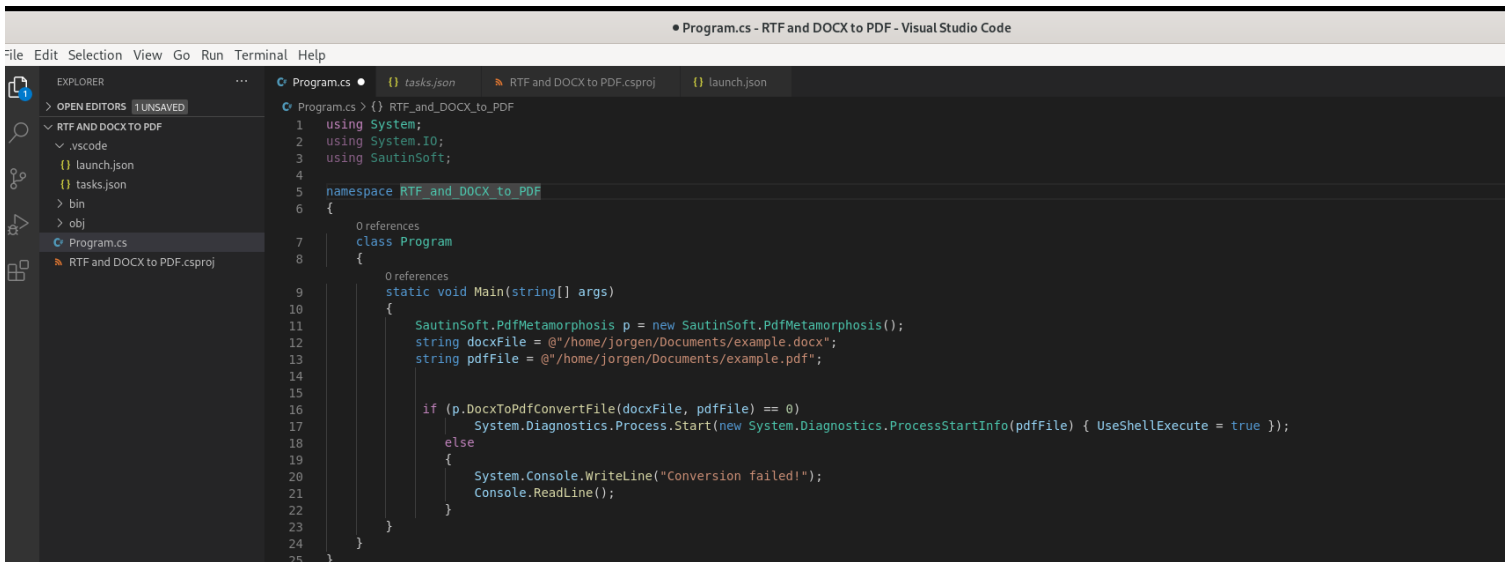


The image is a screenshot of a code editor with four tabs: Program.cs, tasks.json, RTF and DOCX to PDF.csproj (active), and launch.json. The active tab shows the contents of the `RTF and DOCX to PDF.csproj` file. The code is as follows:

```
1 <Project Sdk="Microsoft.NET.Sdk">
2   <PropertyGroup>
3     <OutputType>Exe</OutputType>
4     <TargetFramework>netcoreapp3.1</TargetFramework>
5     <RootNamespace>RTF_and_DOCX_to_PDF</RootNamespace>
6   </PropertyGroup>
7   <ItemGroup>
8     <PackageReference Include="sautinsoft.pdfmetamorphosis" Version="8.3.7.7"/>
9     <PackageReference Include="runtime.osx.10.10-x64.CoreCompat.System.Drawing" Version="5.23.62"/>
10  </ItemGroup>
11 </Project>
```

Good, now our application has the reference to **sautinsoft.pdfmetamorphosis** package and we can write the code to convert rtf and docx to pdf formats.

Follow to the **Explorer**, open the **Program.cs**, remove all the code and type the new:



The new code:

```

using System;
using System.IO;
using SautinSoft;

namespace RTF_and_DOCX_to_PDF
{
    class Program
    {
        static void Main(string[] args)
        {
            SautinSoft.PdfMetamorphosis p = new SautinSoft.PdfMetamorphosis();
            string docxFile = @"/home/jorgen/Documents/example.docx";
            string pdfFile = @"/home/jorgen/Documents/example.pdf";

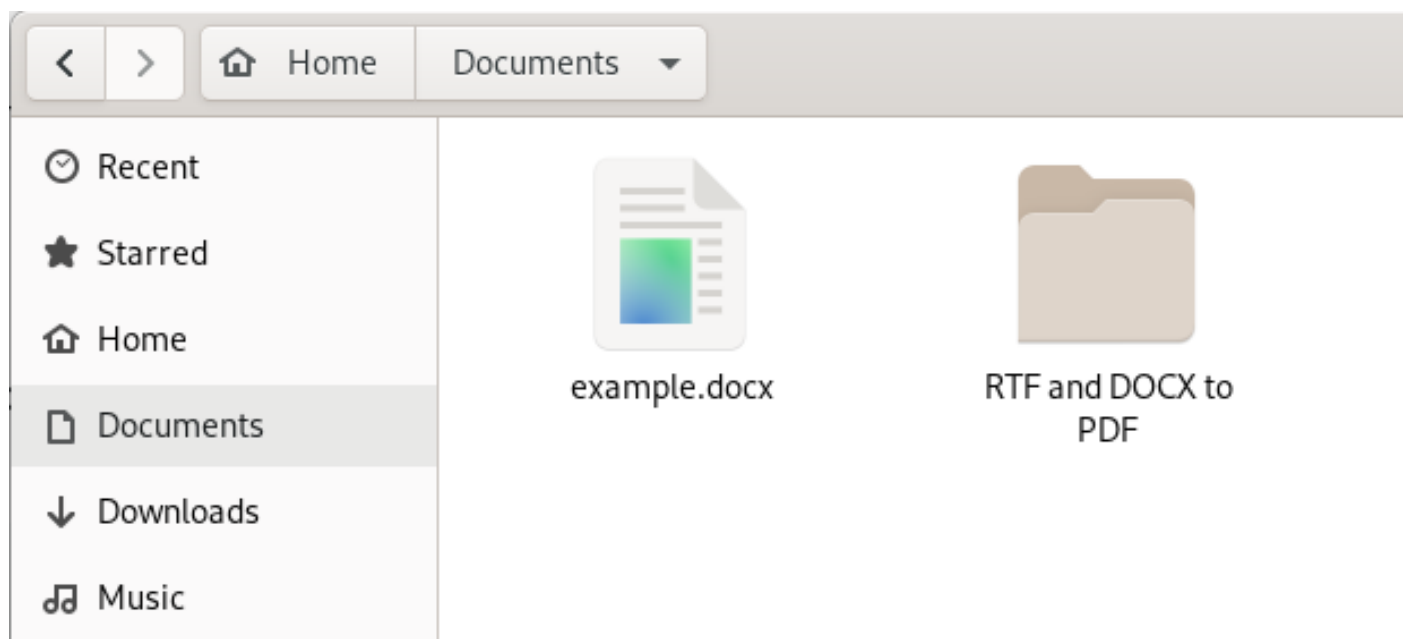
            if (p.DocxToPdfConvertFile(docxFile, pdfFile) == 0)
            System.Diagnostics.Process.Start(new System.Diagnostics.ProcessStartInfo(pdfFile) { UseShellExecute = true });

            else

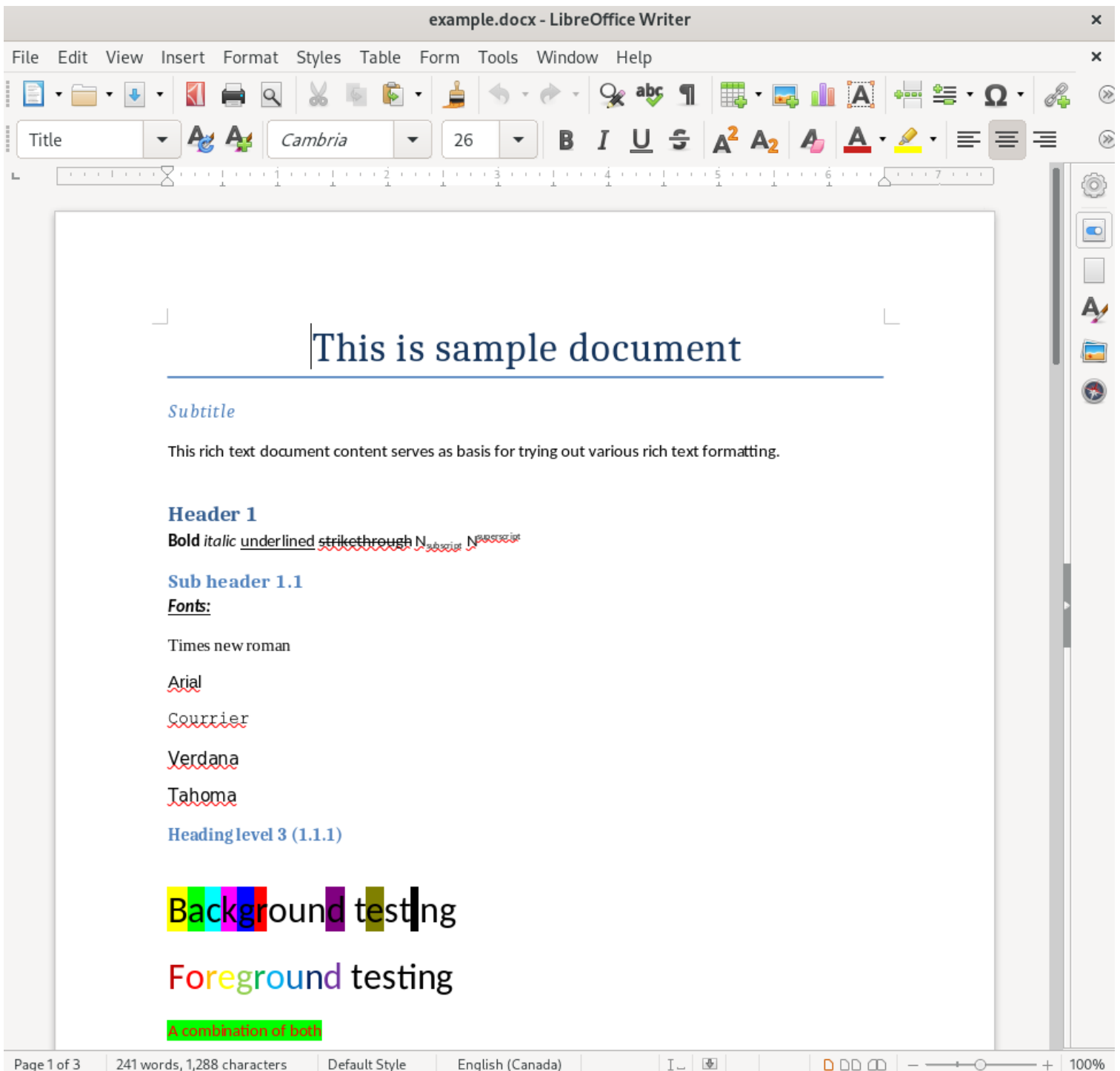
            {
                System.Console.WriteLine("Conversion failed!");
                Console.ReadLine();
            }
        }
    }
}

```

To make tests, we need an input DOCX document. For our tests, let's place the DOCX file with the name "example.docx" at the Documents.

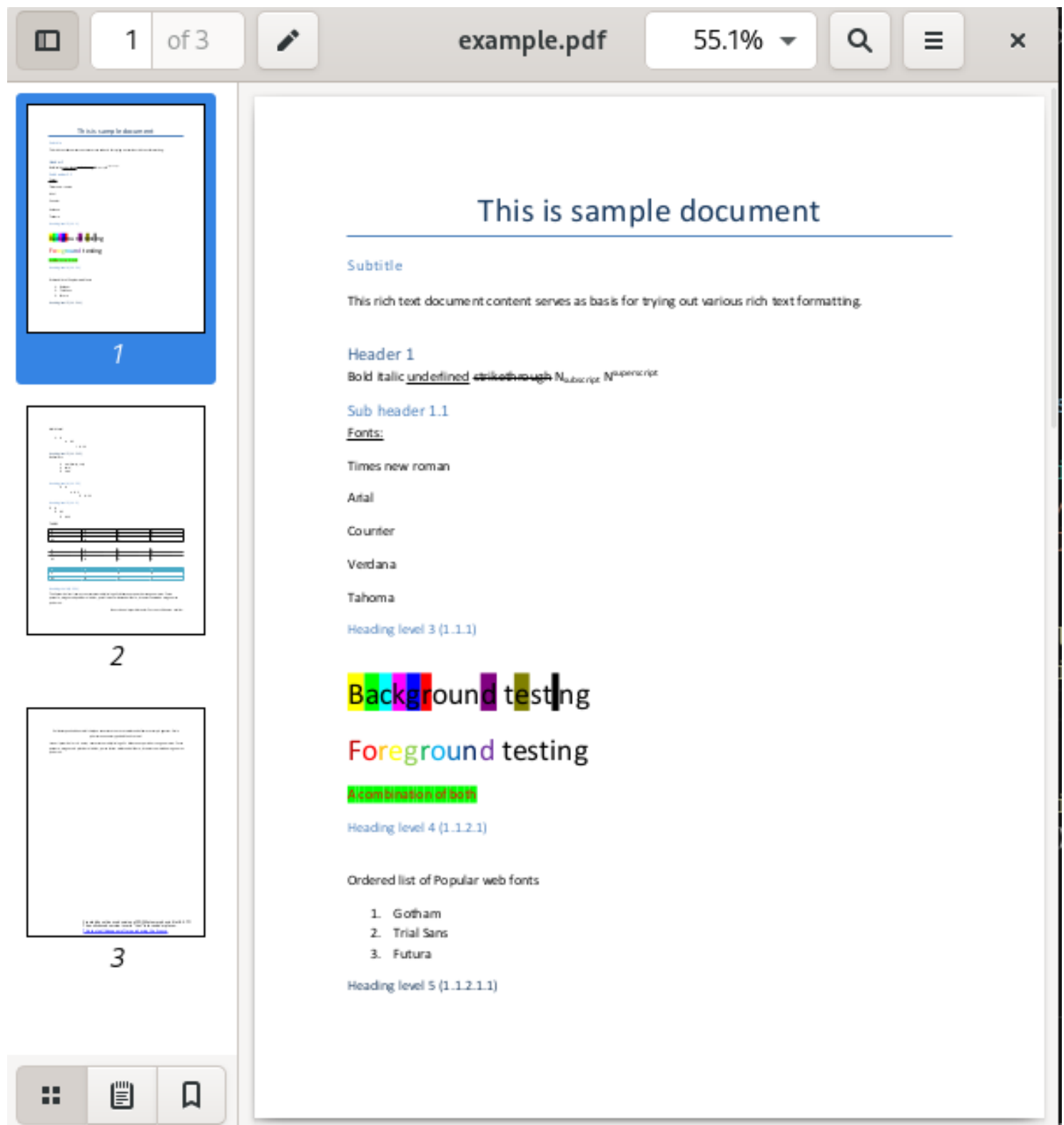


If we open this file in the default Word's Viewer, we'll its contents:

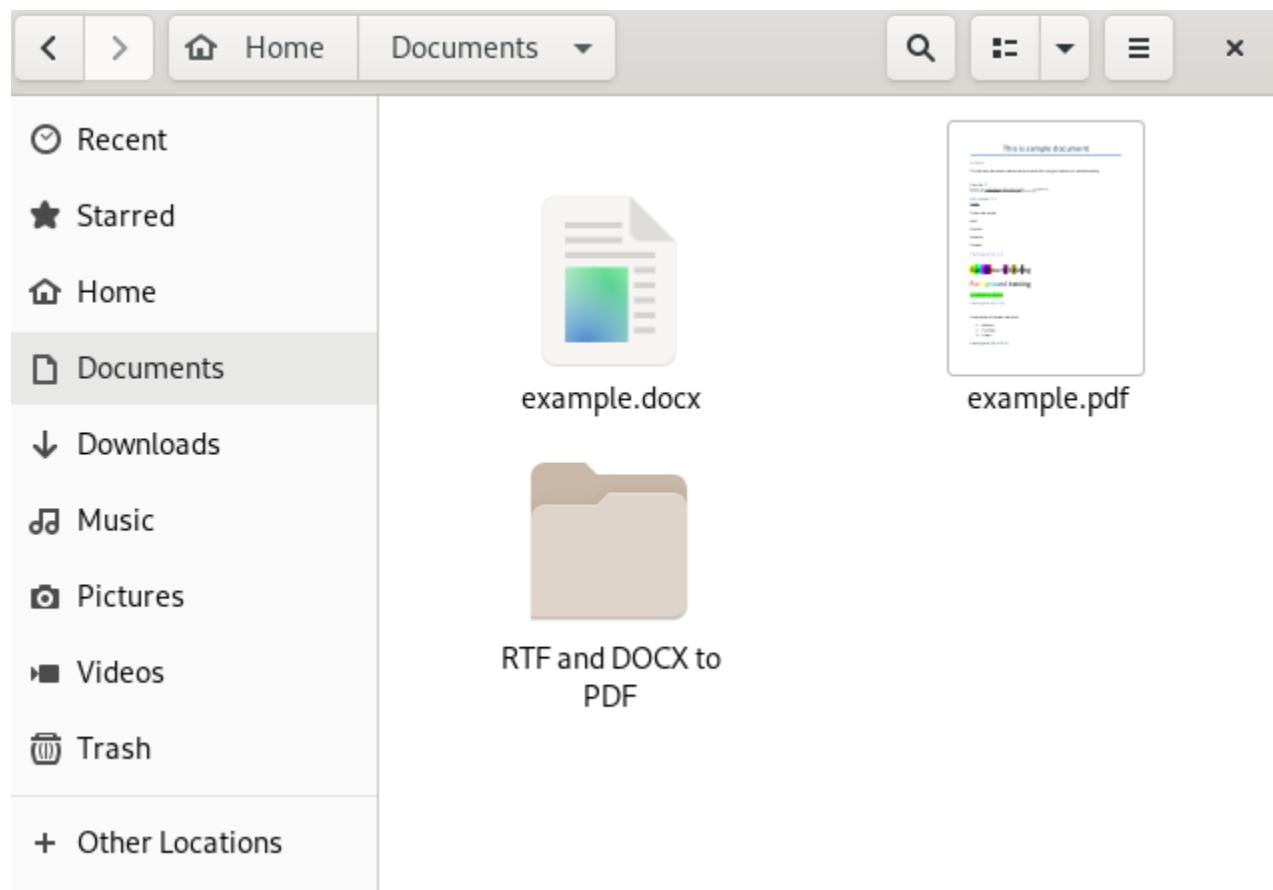


Launch our application and convert the "example.docx" into "example.pdf", type the command: **dotnet run**

You will see the result: "Example.pdf" and if open it :



Files "example.docx" and "example.pdf" have to appear on the Documents:



Well done! You have created the "Word to PDF" application under Linux!

If you have any troubles or need extra code, or help, don't hesitate to ask our SautinSoft Team at support@sautinsoft.com.