RTF to HTML .Net

(Multi-platform .Net library)

SautinSoft

Linux development manual

Table of Contents

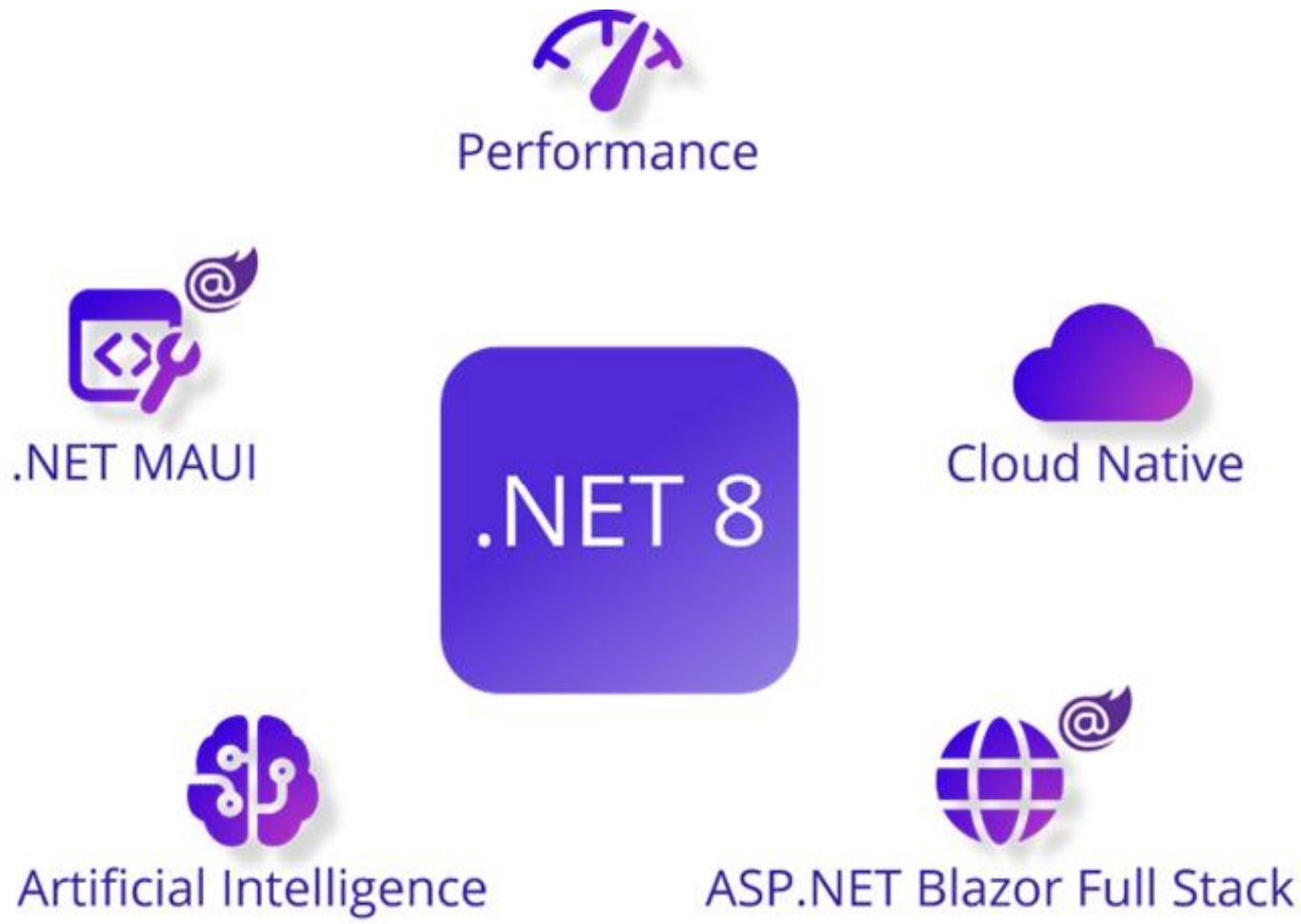
1. Preparing environment

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.



Install VS Code for Linux.

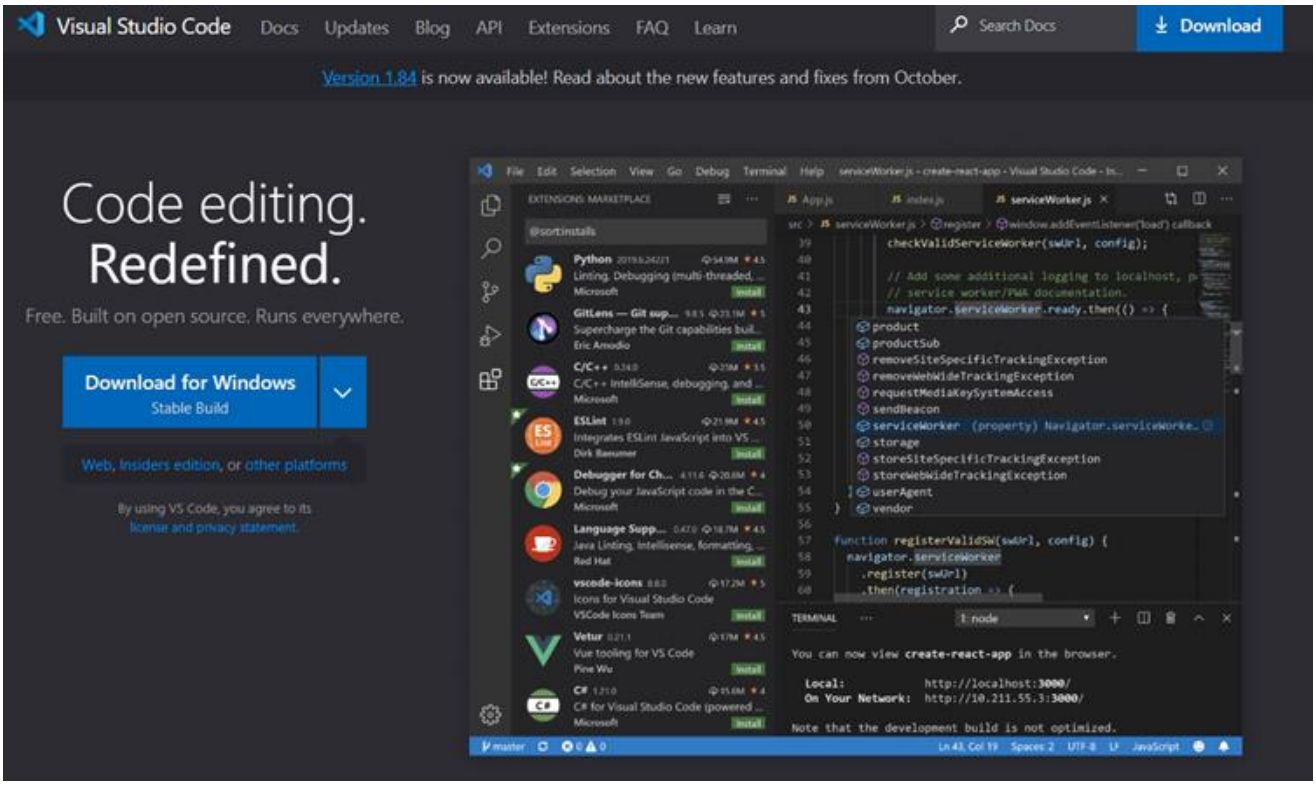
Once installed VS Code, you need to install a C# extension to facilitate us to code and debugging:

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 2017.



Check that the directory with fonts "/usr/share/fonts/truetype" is exist.

Also check that it contains *.ttf files.

If you don't see this folder, make these steps:

1. Download the archive with *.ttf fonts: https://sautinsoft.com/components/fonts.tar
2. Uncompress the downloaded font's archive to a directory and add it to the font path, a list of directories containing fonts:

   ```
   # tar xvzf
   ```

3. Create a directory for new fonts

   ```
   # mkdir /usr/share/fonts/truetype
   ```

4. Move the uncompressed font files to the new font directory

```
# mv *.ttf /usr/share/fonts/truetype
```

5. Navigate to the font directory

```
# cd /usr/share/fonts/truetype
```

6. Create fonts.scale and fonts.dir

```
# mkfontscale && mkfontdir
# fc-cache
```

7. Add the new font directory to the X11 font path

```
# chkfontpath --add /usr/share/fonts/truetype
```

8. Restart X font server

```
# /etc/rc.d/init.d/xfs restart
```

You can verify the successful addition of the new path by running `chkfontpath` command or by listing X font server's /etc/X11/XF86Config file.

If you do not have root access, copy the *.ttf to ~/.fonts directory instead.

Or you may install “Microsoft TrueType core fonts” using terminal and command:

```
$ sudo apt install ttf-mscorefonts-installer
```

```
linuxconfig@linuxconfig-org: ~

All done, no errors.
Extracting cabinet: /var/lib/update-notifier/package-data-downloads/partial/verdan32.exe
  extracting fontinst.exe
  extracting fontinst.inf
  extracting Verdanab.TTF
  extracting Verdanai.TTF
  extracting Verdanaz.TTF
  extracting Verdana.TTF

All done, no errors.
Extracting cabinet: /var/lib/update-notifier/package-data-downloads/partial/webdin32.exe
  extracting fontinst.exe
  extracting Webdings.TTF
  extracting fontinst.inf
  extracting Licen.TXT

All done, no errors.
All fonts downloaded and installed.
Processing triggers for man-db (2.9.0-2) ...
Processing triggers for fontconfig (2.13.1-2ubuntu2) ...
linuxconfig@linuxconfig-org:~$
```

Read more about [TrueType Fonts and "How to install Microsoft fonts, How to update fonts cache files, How to confirm new fonts installation"](#) .

With these steps, we will ready to start developing.

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

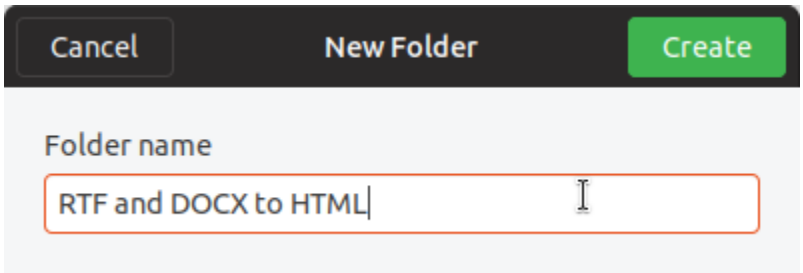
[Get Started with C# and Visual Studio Code](#)

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

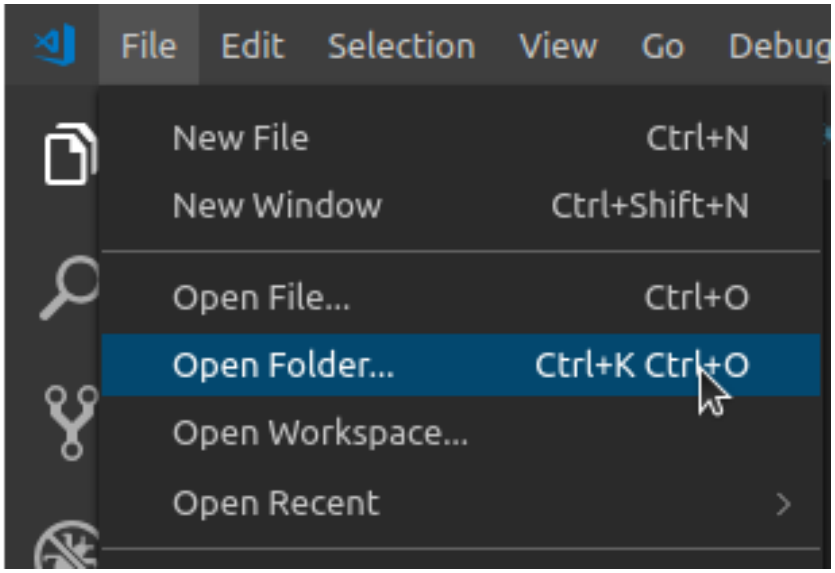
2. Creating “Convert RTF/DOCX to HTML” app

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

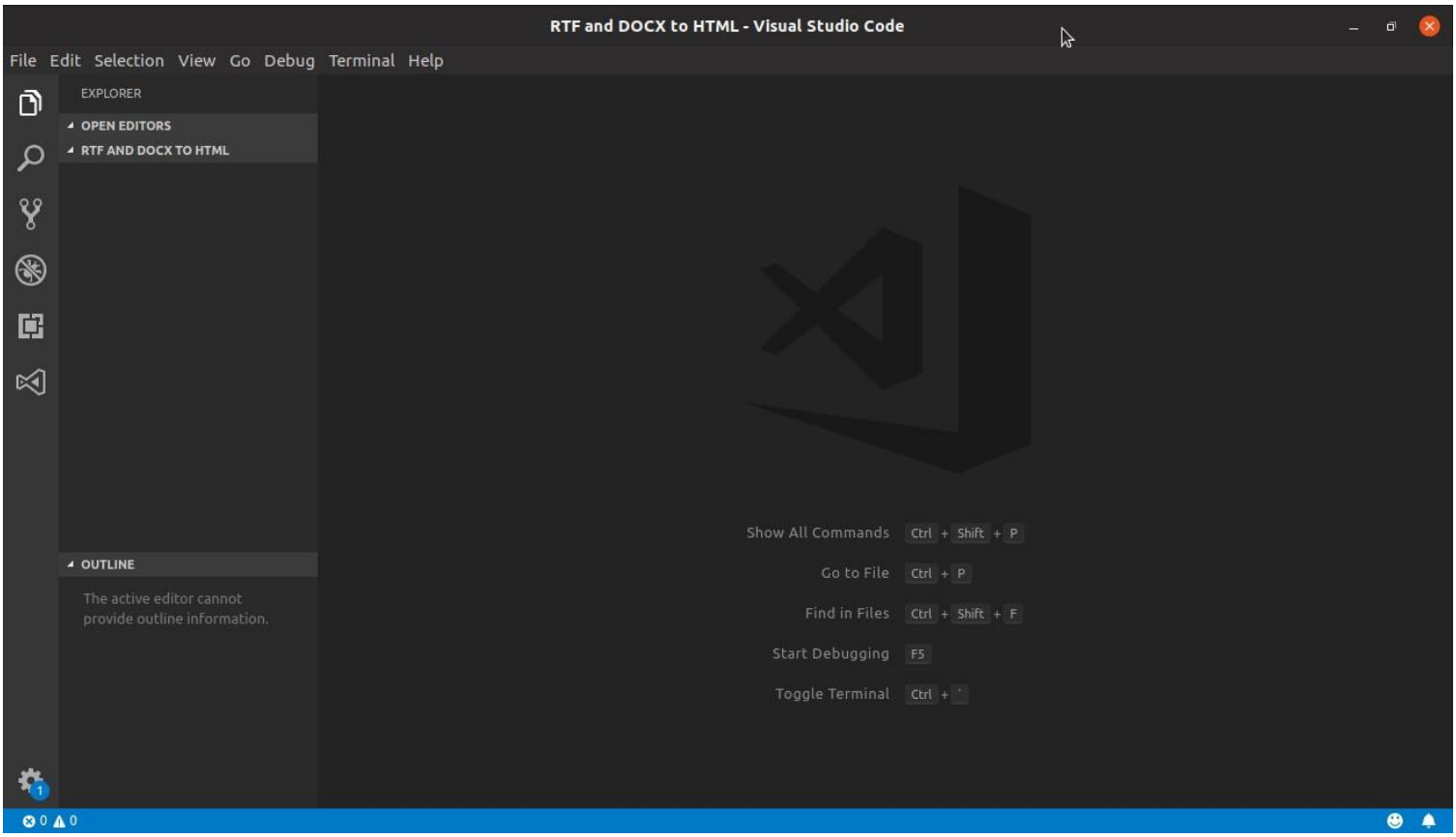
For example, let’s create the folder “***RTF and DOCX to HTML***” on the Desktop (Right click-> New Folder)



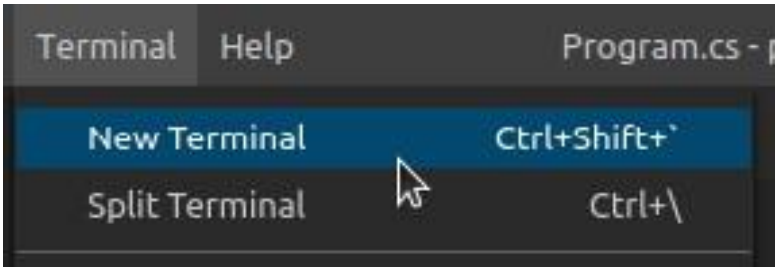
Open VS Code and click in the menu ***File->Open Folder***. From the dialog, open the folder you’ve created previously:



Next you will see the similar screen:

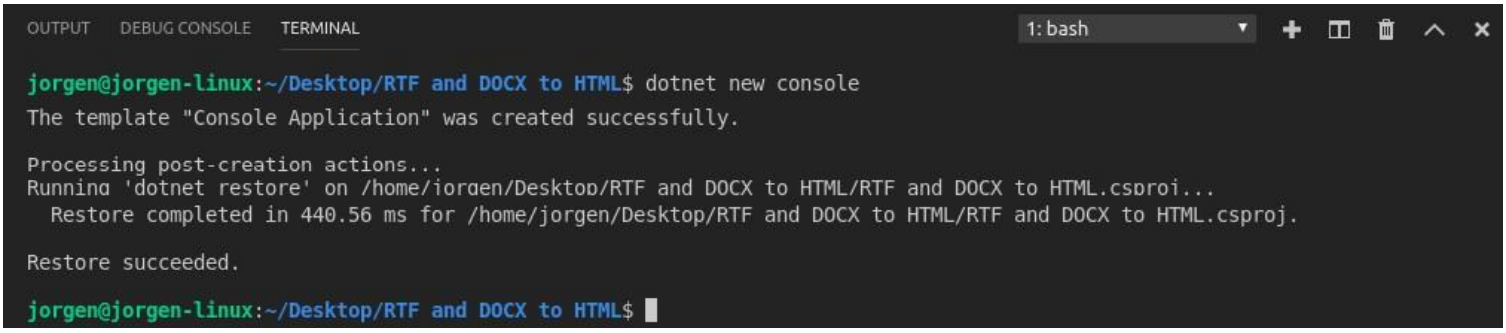


Now, open the integrated console – the Terminal: follow to the menu ***Terminal -> New Terminal*** (or press Ctrl+Shift+'):



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***

```
OUTPUT   DEBUG CONSOLE   TERMINAL                                   1: bash

jorgen@jorgen-linux:~/Desktop/RTF and DOCX to HTML$ dotnet run
Hello World!
jorgen@jorgen-linux:~/Desktop/RTF and DOCX to HTML$
```

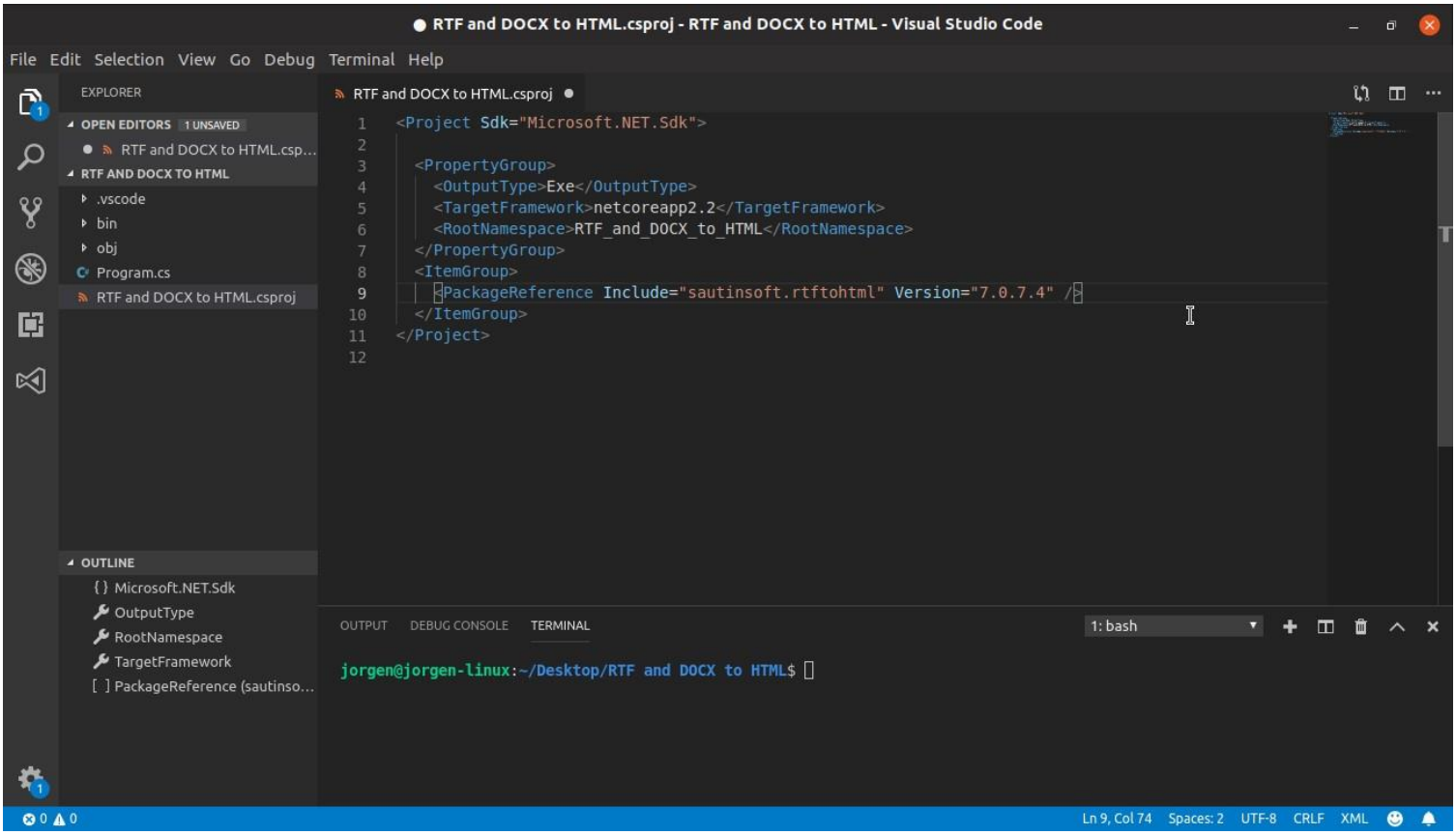
You can see the typical "Hello world!" message.

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

We'll transform it into an application that will convert rtf and docx files into HTML format.

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

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



Add these lines into the file ***"RTF and DOCX to HTML.csproj"***:

```
<ItemGroup>
    <PackageReference Include="sautinsoft.rtftohtml" Version="2024.X.X"
    />
```

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

At the moment of writing this manual, the latest version of ***sautinsoft.rtftohtml*** was 2024.3.1. But you may specify the latest version, to know what is the latest, follow:

https://www.nuget.org/packages/sautinsoft.rtftohtml/

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

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

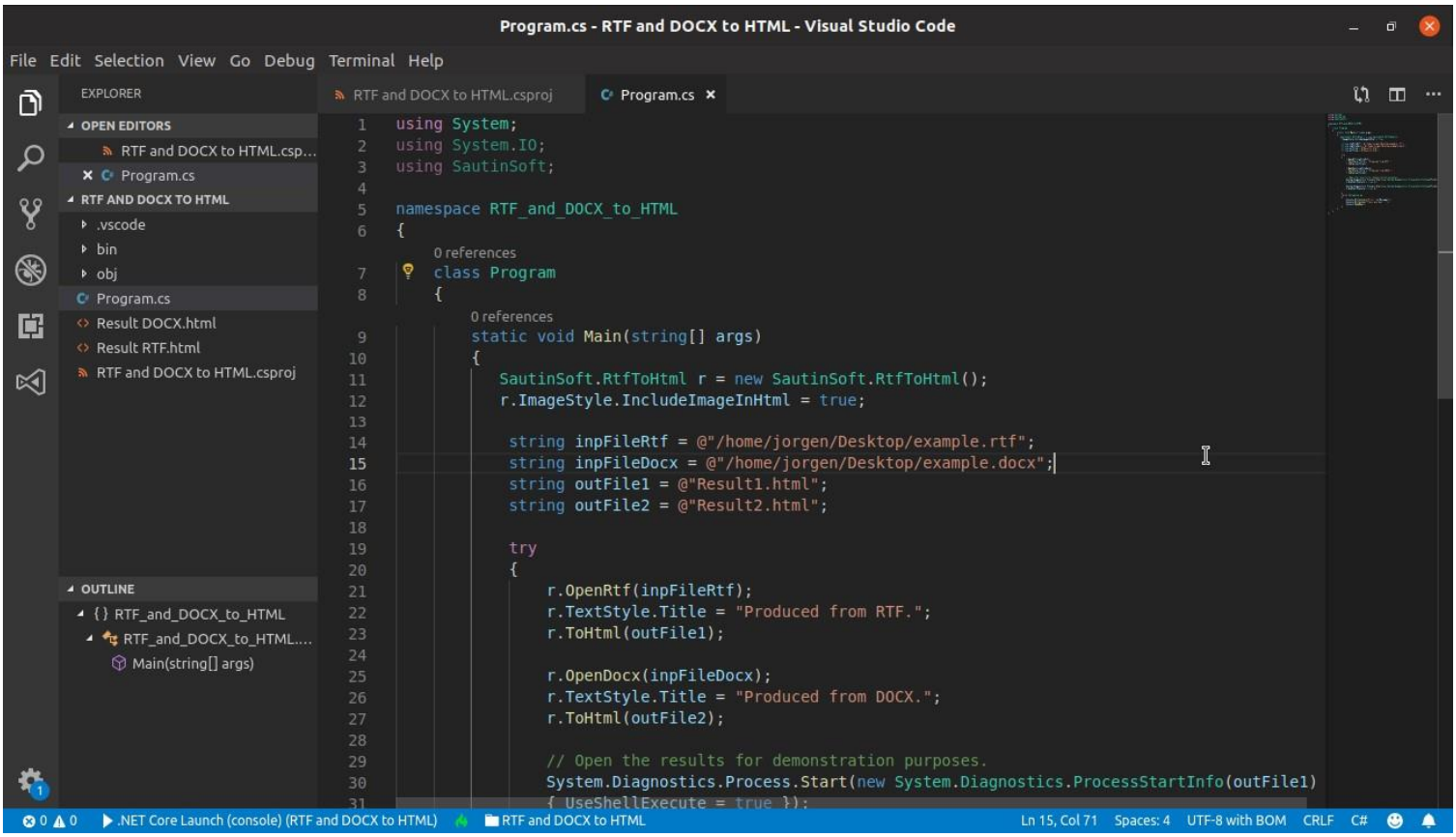
```
OUTPUT   DEBUG CONSOLE   TERMINAL                                   2: bash
jorgen@jorgen-linux:~/Desktop/RTF and DOCX to HTML$ dotnet restore
  Restore completed in 157.2 ms for /home/jorgen/Desktop/RTF and DOCX to HTML/RTF and DOCX to HTML.csproj.
jorgen@jorgen-linux:~/Desktop/RTF and DOCX to HTML$
```

Good, now our application has the reference to ***sautinsoft.rtftohtml*** package and we can write the code to convert DOCX and RTF documents into HTML format.

Add these lines into your project also:

```
<ItemGroup>
<PackageReference Include="Svg.Skia" Version="1.0.0.18" />
<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" Version="2.88.7" />
<PackageReference Include="SkiaSharp.NativeAssets.Linux" Version="2.88.7" />
</ItemGroup>
```

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



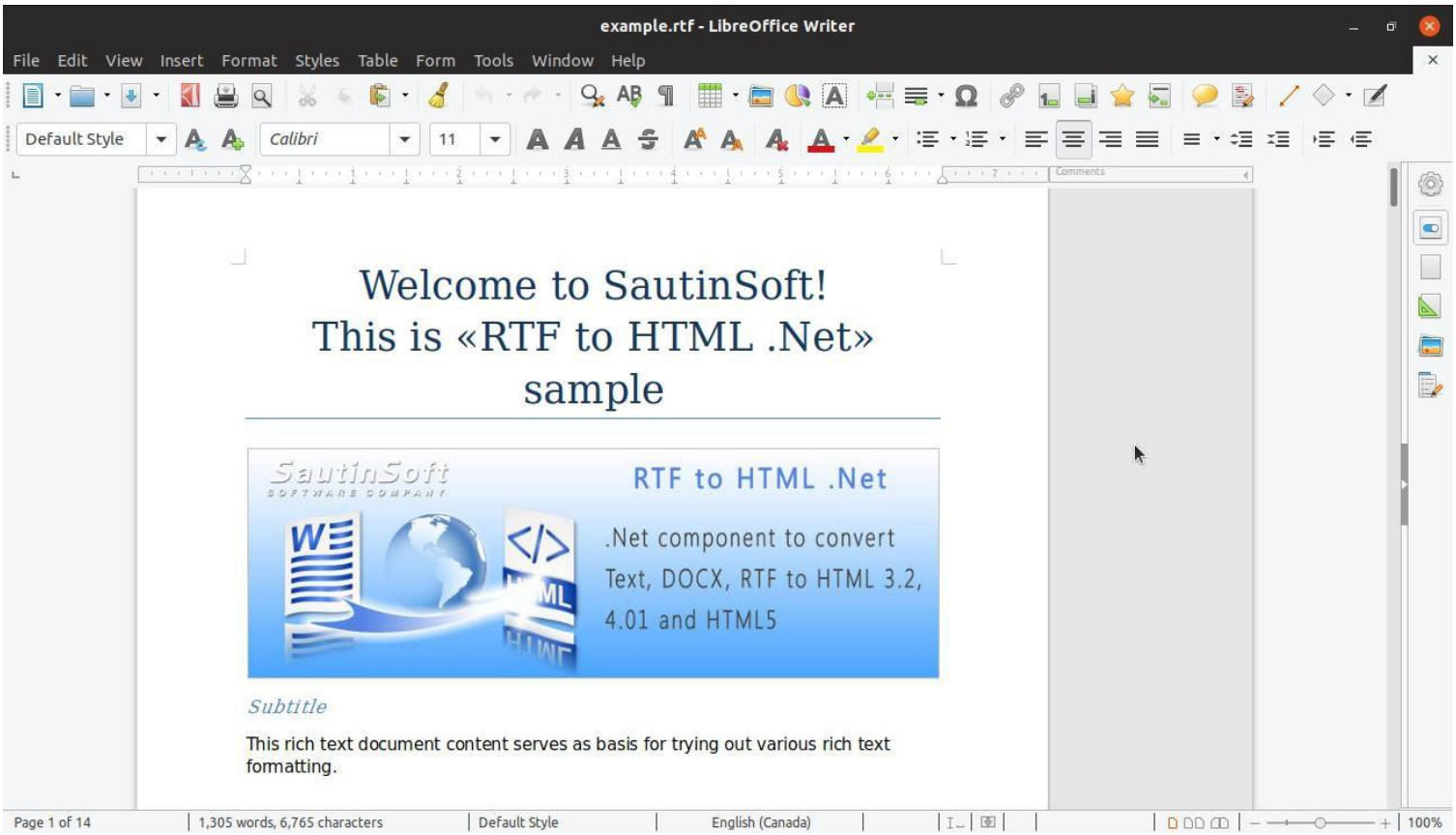
The new code:

```
using System;
using System.IO;
using SautinSoft;

namespace RTF_and_DOCX_to_HTML
{
    class Program
    {
        static void Main(string[] args)
        {
           SautinSoft.RtfToHtml r = new SautinSoft.RtfToHtml();
           r.ImageStyle.IncludeImageInHtml = true;

            string inpFileRtf = @"/home/jorgen/Desktop/example.rtf";
            string inpFileDocx = @"/home/jorgen/Desktop/example.docx";
            string outFile1 = @"Result1.html";
            string outFile2 = @"Result2.html";

            try
            {
                r.OpenRtf(inpFileRtf);
                r.TextStyle.Title = "Produced from RTF.";
                r.ToHtml(outFile1);

                r.OpenDocx(inpFileDocx);
                r.TextStyle.Title = "Produced from DOCX.";
                r.ToHtml(outFile2);

                // Open the results for demonstration purposes.
                System.Diagnostics.Process.Start(new System.Diagnostics.ProcessStartInfo(outFile1)
                { UseShellExecute = true });
```

```
                System.Diagnostics.Process.Start(new System.Diagnostics.ProcessStartInfo(outFile2)
                { UseShellExecute = true });

            }
            catch (Exception e)
            {
                Console.WriteLine($"Error: {e.Message}");
                Console.WriteLine("Press any key ...");
                Console.ReadKey();
            }
        }
    }
}
```

To make tests, we need the input RTF and DOCX documents. For our tests, let's place the files "example.rtf" and "example.docx" at the Desktop.

If we open these files in the default viewer, we'll see their content:

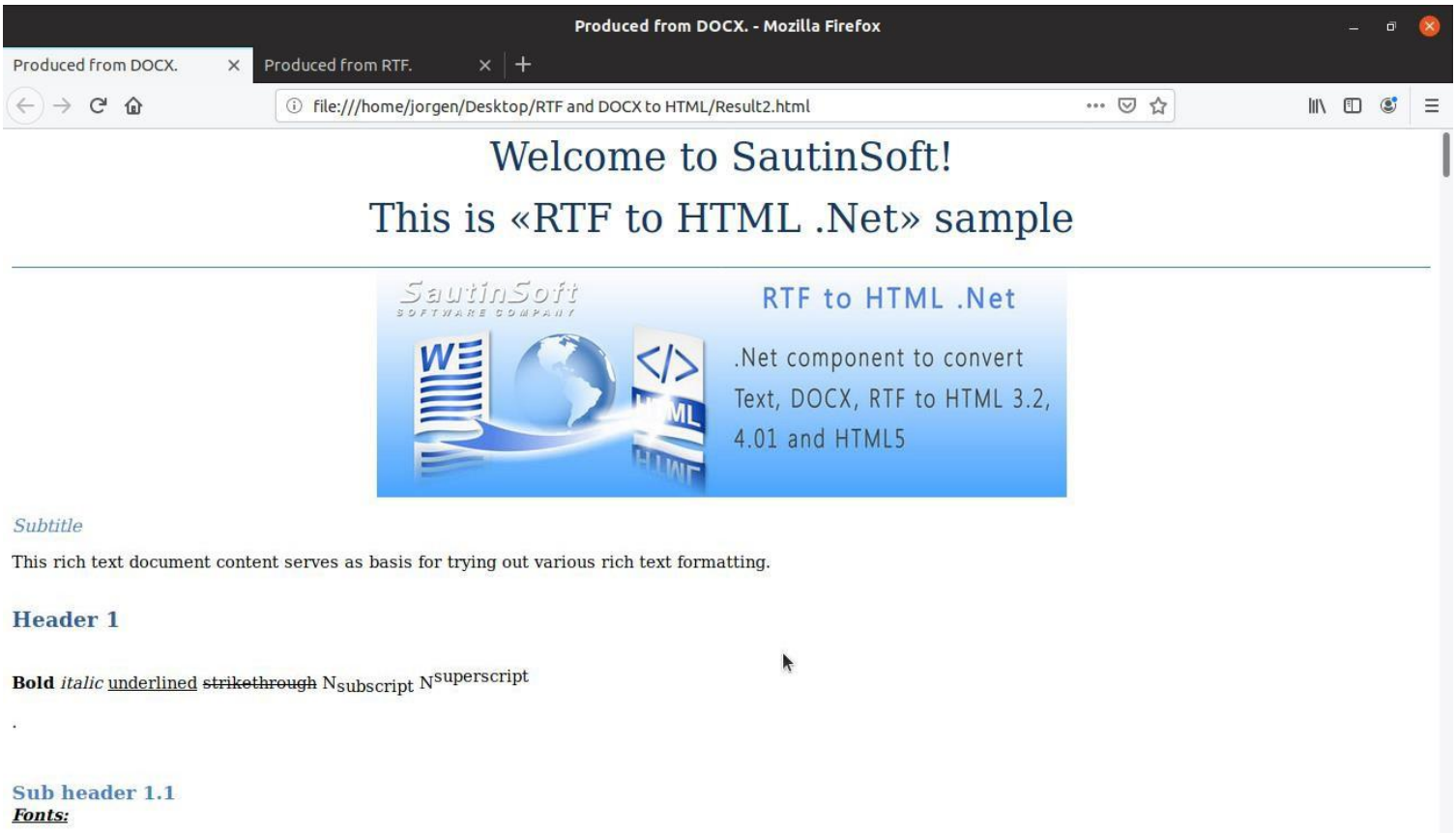


Launch our application and convert the “example.rtf” and “example.docx” into HTML documents, type the command: ***dotnet run***

```
OUTPUT    DEBUG CONSOLE    TERMINAL                                  1: bash
jorgen@jorgen-linux:~/Desktop/RTF and DOCX to HTML$ dotnct run
```

If you see the opening browser with the output HTML documents, everything is fine and we can check the results produced by the [RTF to HTML .Net]() library.

The new files “sample.rtf” and “sample.docx” have to appear on the Desktop:



Well done! You have created the “RTF/DOCX to HTML” 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]().