RTF to HTML .Net

(Multi-platform .Net library)

SautinSoft

Linux development manual

Table of Contents

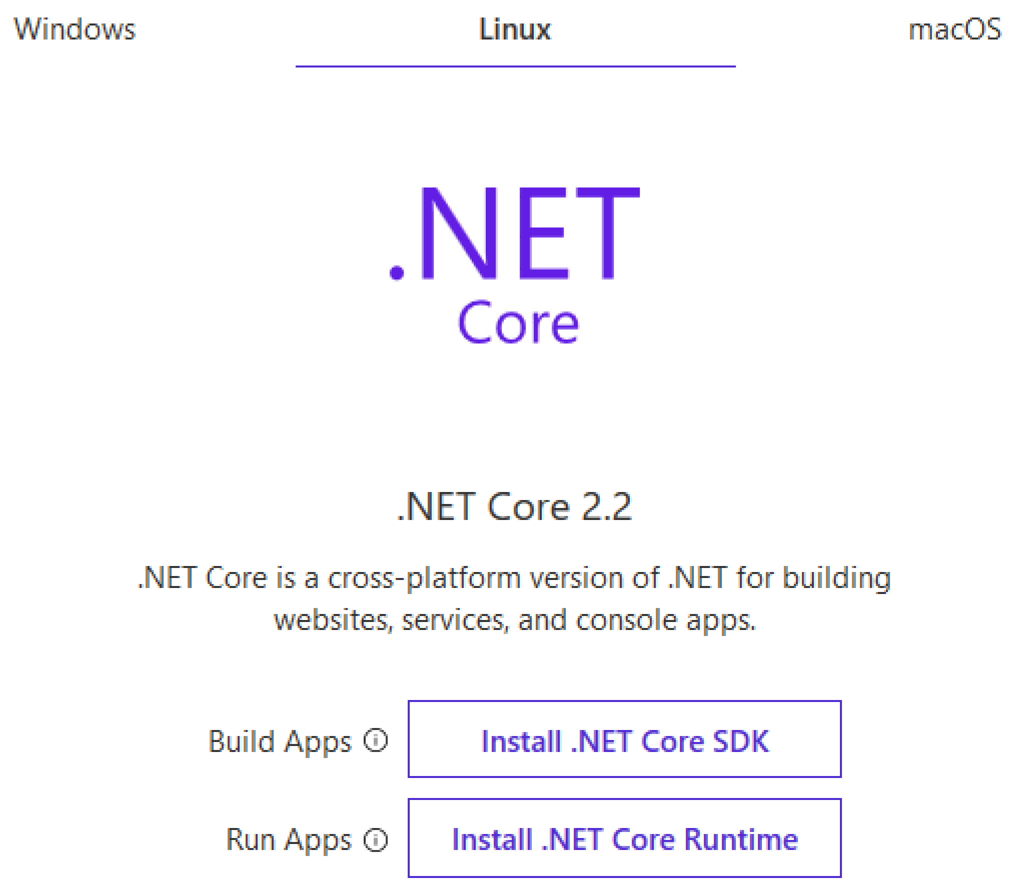
1. Preparing environment

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.



.NET Core 2.2

.NET Core is a cross-platform version of .NET for building websites, services, and console apps.

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:

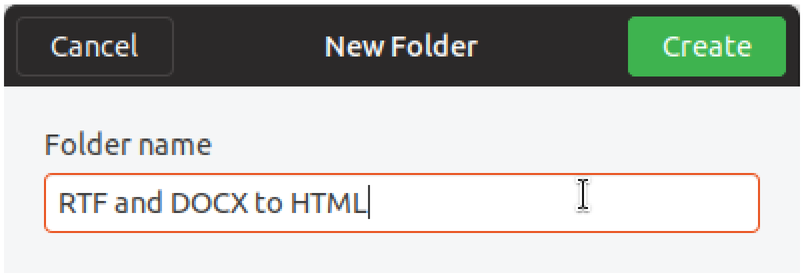
Get Started with C# and Visual Studio Code

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.

2. Creating "Convert RTF/DOCX to HTML" app

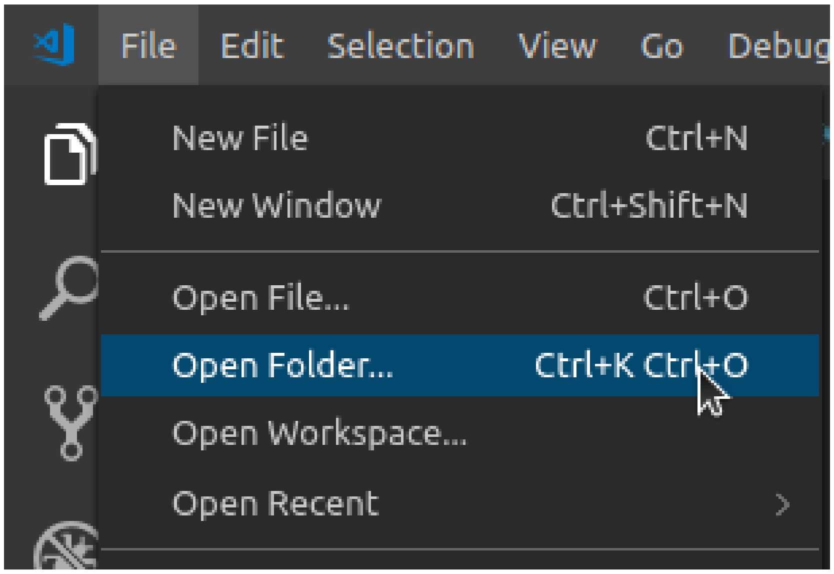
Create a new folder in your Linux machine with the name ***RTF and DOCX to HTML.***

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

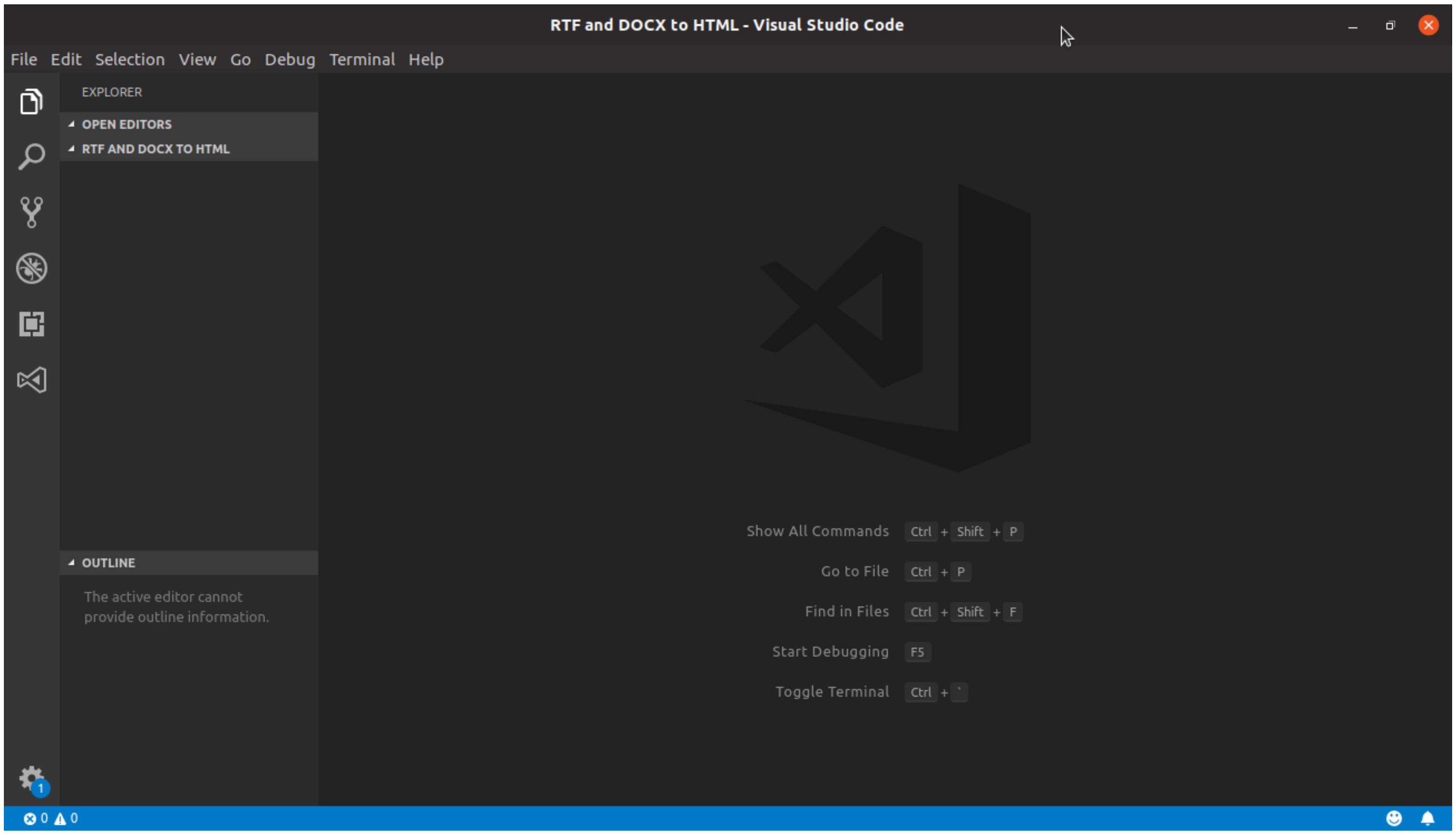


RTF and DOCX to HTML

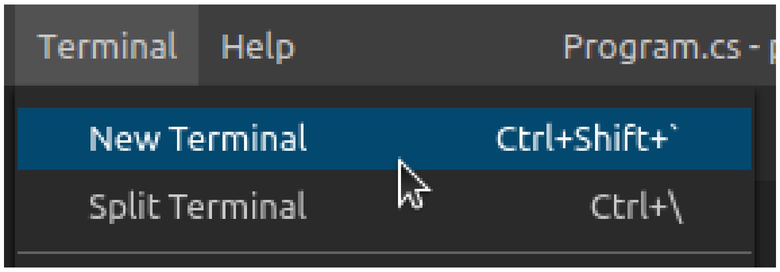
Open VS Code and click in the menu ***File->Open Folder***. From the dialog, open the folder you've created previously:



Next you will see the similar screen:

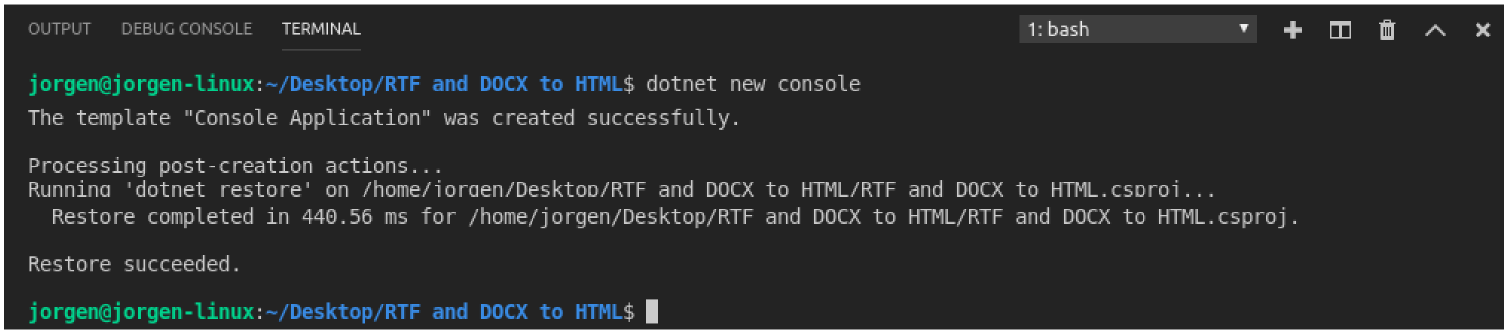

RTF and DOCX to HTML - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
RTF AND DOCX TO HTML
OUTLINE
The active editor cannot provide outline information.
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Start Debugging F5
Toggle Terminal Ctrl + `

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



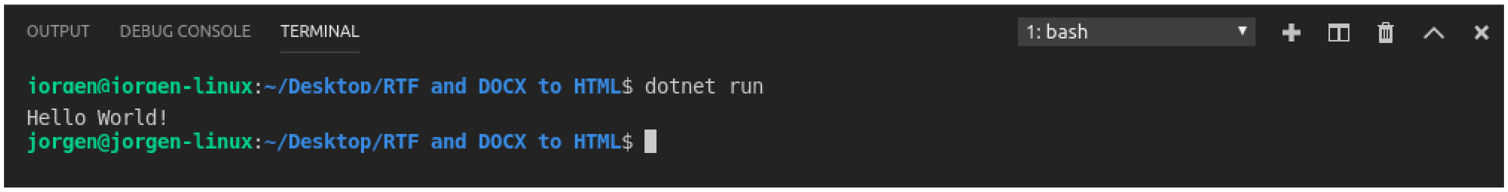
Create a new console application, using ***dotnet*** command.

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



```
jorgen@jorgen-linux:~/Desktop/RTF and DOCX to HTML$ dotnet new console
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on /home/jorgen/Desktop/RTF and DOCX to HTML/RTF and DOCX to HTML.csproj...
  Restore completed in 440.56 ms for /home/jorgen/Desktop/RTF and DOCX to HTML/RTF and DOCX to HTML.csproj.

Restore succeeded.

jorgen@jorgen-linux:~/Desktop/RTF and DOCX to HTML$
```

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



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

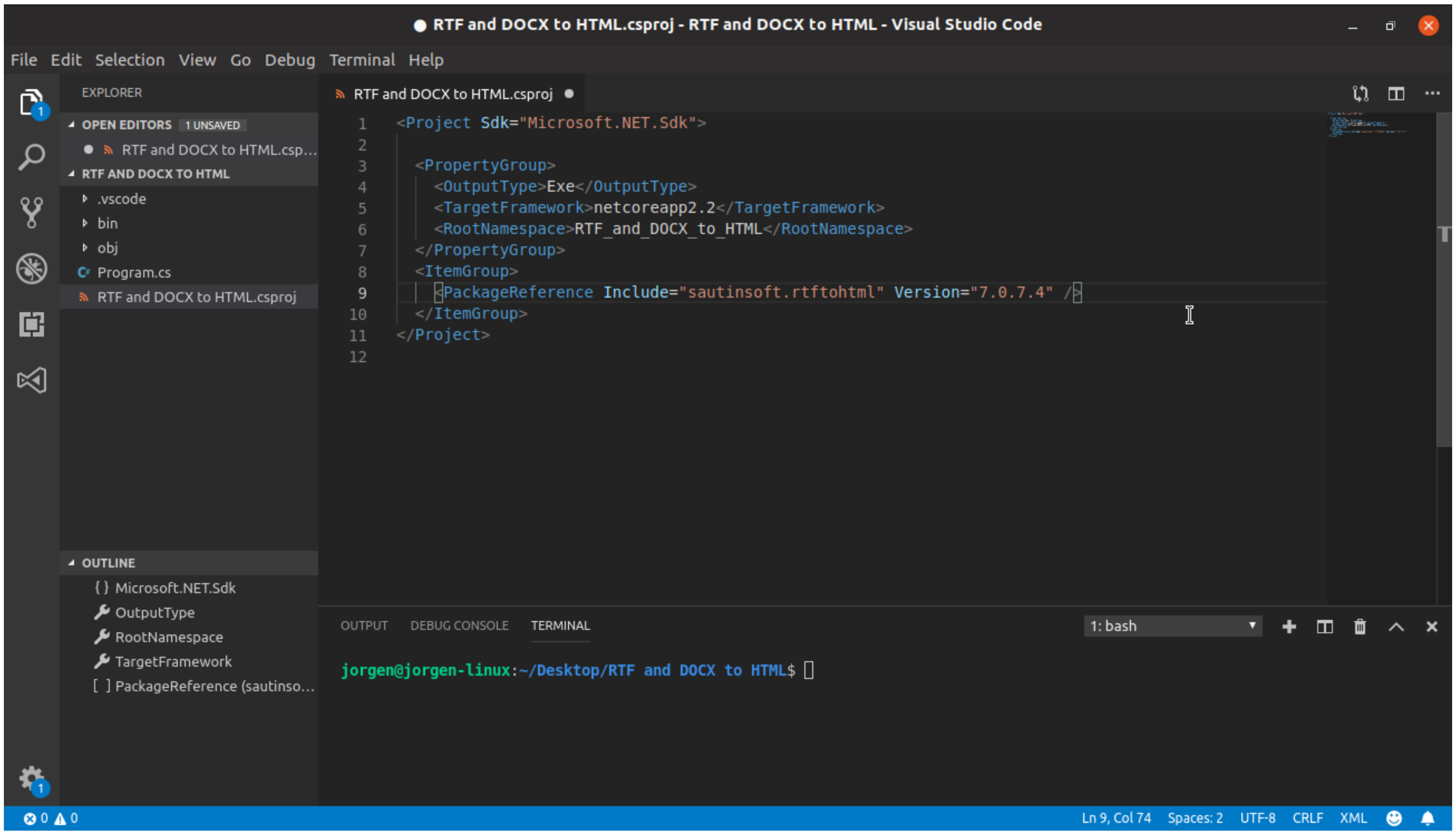
You can see the typical “Hello world!” message.

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:



● RTF and DOCX to HTML.csproj - RTF and DOCX to HTML - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS 1 UNSAVED
RTF and DOCX to HTML.csp...
RTF AND DOCX TO HTML
.vscode
bin
obj
Program.cs
RTF and DOCX to HTML.csproj
OUTLINE
{ } Microsoft.NET.Sdk
OutputType
RootNamespace
TargetFramework
[ ] PackageReference (sautinso...
RTF and DOCX to HTML.csproj ●
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
⊗ 0 ⚠ 0
Ln 9, Col 74
Spaces: 2
UTF-8
CRLF
XML

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.2</TargetFramework>
    <RootNamespace>RTF_and_DOCX_to_HTML</RootNamespace>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="sautinsoft.rtftohtml" Version="7.0.7.4" />
  </ItemGroup>
</Project>
```

```
jorgen@jorgen-linux:~/Desktop/RTF and DOCX to HTML$
```

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

```
<ItemGroup>
     <PackageReference Include=”sautinsoft.rtftohtml” Version=”7.0.7.4” />
```

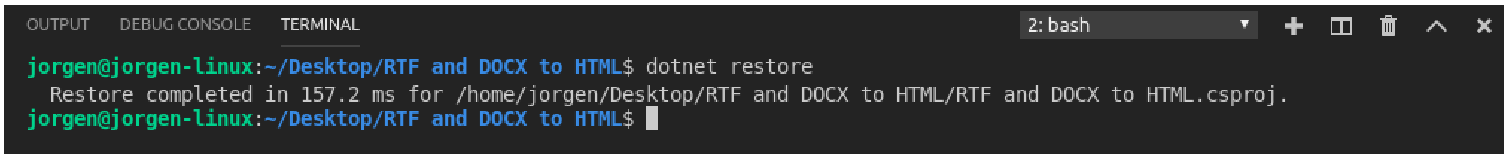
```
</ItemGroup>
```

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

At the moment of writing this manual, the latest version of ***sautinsoft.rtftohtml*** was 7.0.7.4. 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.

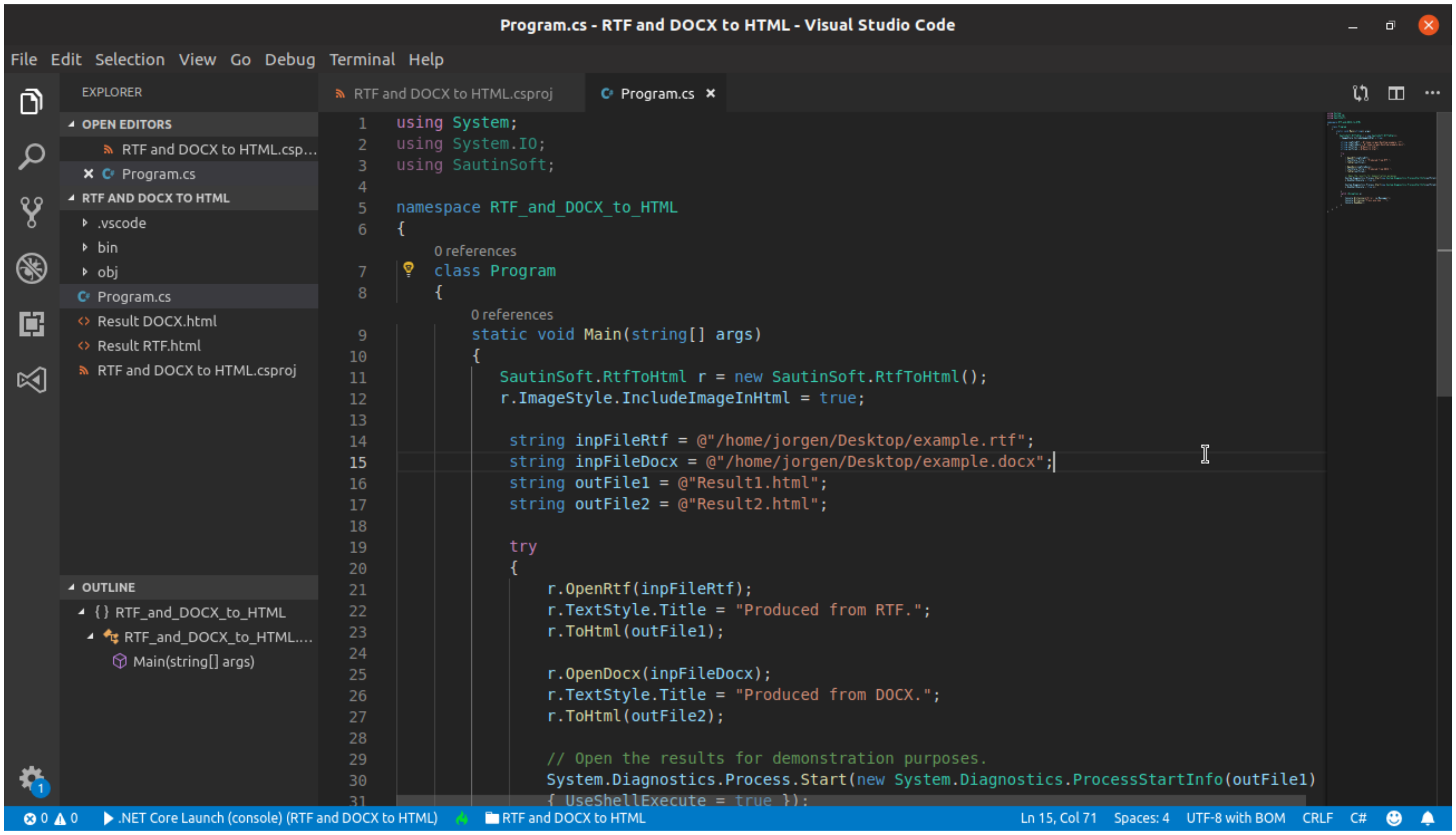
Follow to the ***Terminal*** and type the command: ***dotnet restore***



```
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.

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



Program.cs - RTF and DOCX to HTML - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
EXPLORER
RTF and DOCX to HTML.csproj
Program.cs
OPEN EDITORS
RTF and DOCX to HTML.csp...
Program.cs
RTF AND DOCX TO HTML
.vscode
bin
obj
Program.cs
Result DOCX.html
Result RTF.html
RTF and DOCX to HTML.csproj
OUTLINE
{} RTF_and_DOCX_to_HTML
RTF_and_DOCX_to_HTML....
Main(string[] args)
0 ⚠ 0
.NET Core Launch (console) (RTF and DOCX to HTML)
RTF and DOCX to HTML
Ln 15, Col 71 Spaces: 4 UTF-8 with BOM CRLF C#

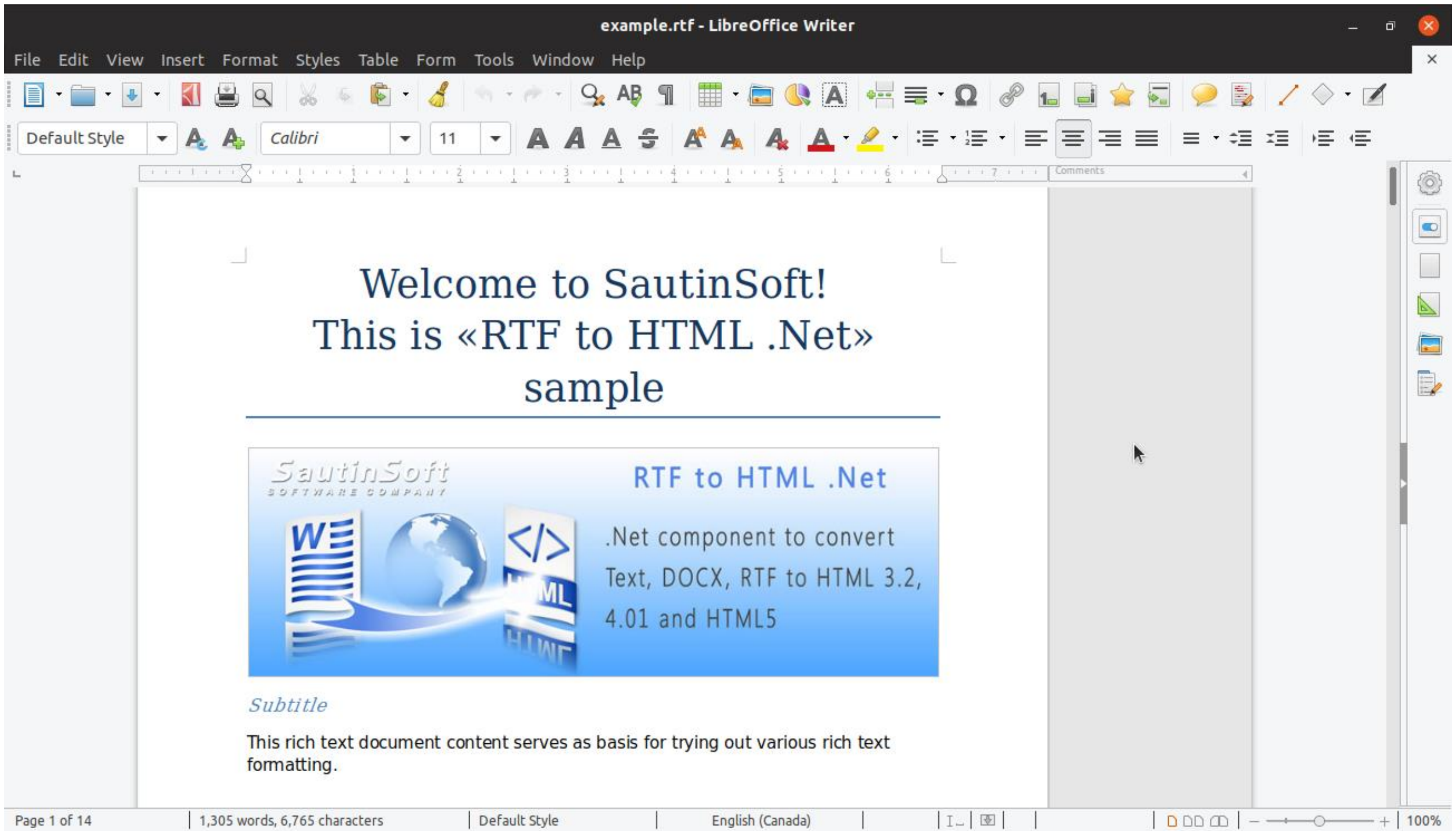
```
using System;
using System.IO;
using SautinSoft;

namespace RTF_and_DOCX_to_HTML
{
    0 references
    class Program
    {
        0 references
        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 });
```

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:


example.rtf - LibreOffice Writer
File Edit View Insert Format Styles Table Form Tools Window Help
Default Style
Calibri
11
SautinSoft SOFTWARE COMPANY
HTML
Page 1 of 14
1,305 words, 6,765 characters
Default Style
English (Canada)
100%

Welcome to SautinSoft!
This is «RTF to HTML .Net» sample

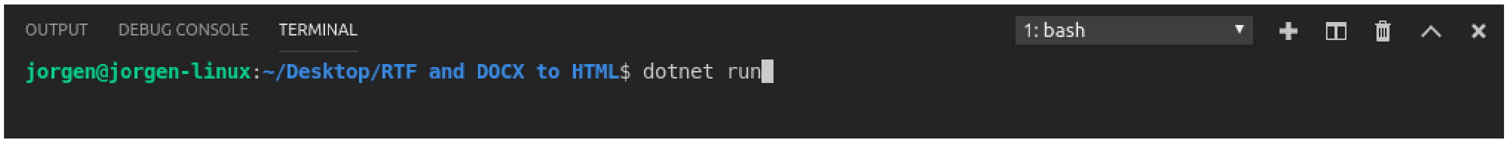
RTF to HTML .Net

.Net component to convert Text, DOCX, RTF to HTML 3.2, 4.01 and HTML5

Subtitle

This rich text document content serves as basis for trying out various rich text formatting.

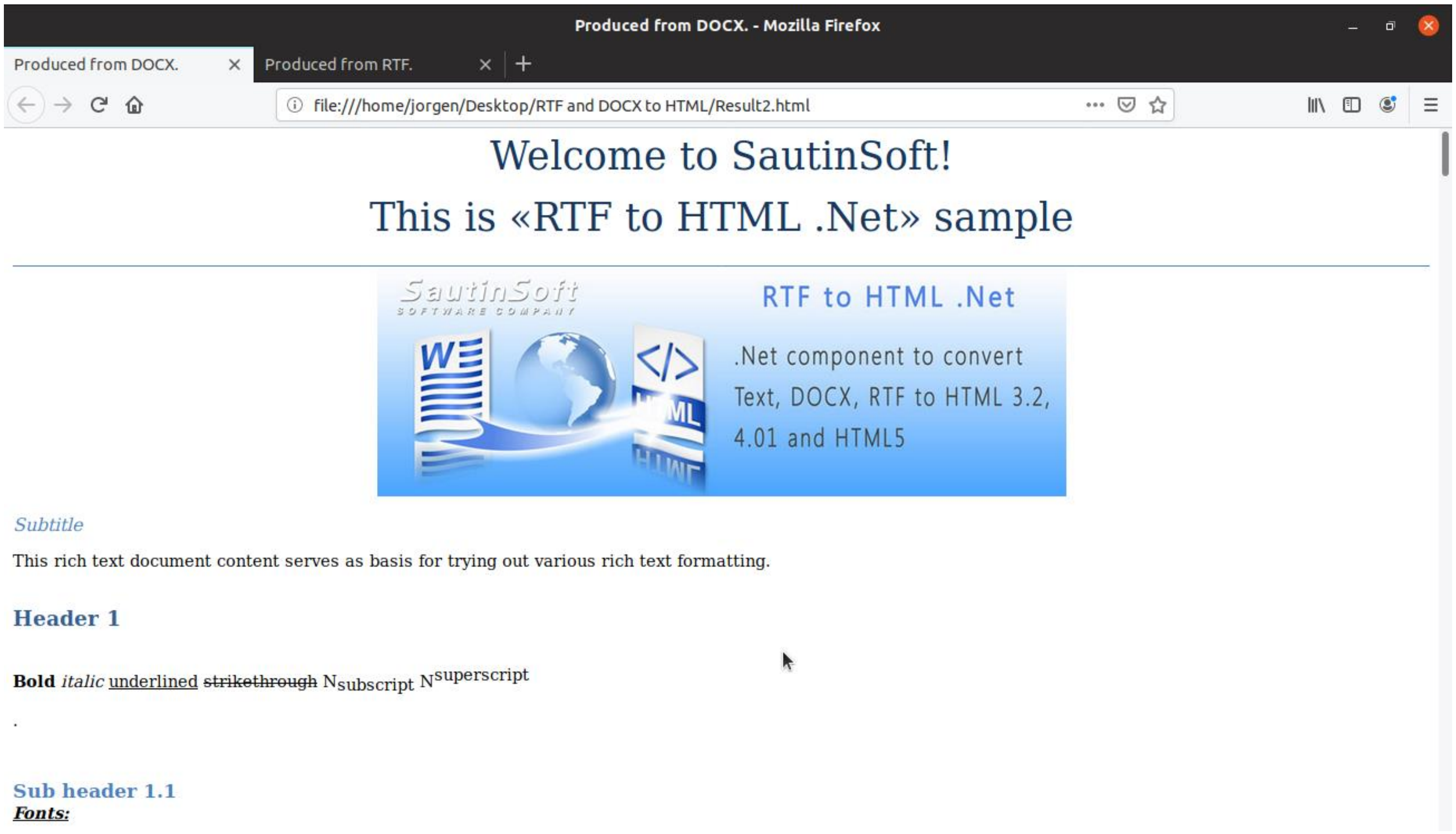
Launch our application and convert the “example.rtf” and “example.docx” into HTML documents, type the command: ***dotnet run***



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


Produced from DOCX. - Mozilla Firefox
Produced from DOCX.
Produced from RTF.
file:///home/jorgen/Desktop/RTF and DOCX to HTML/Result2.html
SautinSoft
SOFTWARE COMPANY
RTF to HTML .Net
.Net component to convert Text, DOCX, RTF to HTML 3.2, 4.01 and HTML5

Welcome to SautinSoft!

This is «RTF to HTML .Net» sample

Subtitle

This rich text document content serves as basis for trying out various rich text formatting.

Header 1

Bold *italic* underlined ~~strikethrough~~ N$_{\text{subscript}}$ N$^{\text{superscript}}$

.

Sub header 1.1

Fonts:

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.