

SautinSoft Document.Net multi-platform .NET library

Linux development manual

Table of Contents

1. Preparing environment.....	2
2. Convert pdf to docx file.....	3
3. Create docx from scratch.....	12
4. Add content to docx file.....	12

Illustration Index

Illustration 1: Explorer.....	3
Illustration 2: Showing folder Convert_pdf_to_docx.....	3
Illustration 3: Integrated console.....	4
Illustration 4: Creating a new console project.....	4
Illustration 5: Running project.....	4
Illustration 6: Editing project configuration file.....	5
Illustration 7: Adding SaintSoft Document dll reference.....	5
Illustration 8: Adding all needed dependencies.....	6
Illustration 9: Adding assets.....	6
Illustration 10: Restore output.....	6
Illustration 11: Editing Program.cs.....	7
Illustration 12: Inserting new code.....	9
Illustration 13: Add pdf file to convert.....	10
Illustration 14: Run command.....	10
Illustration 15: Publish program.....	11
Illustration 16: Native files.....	11
Illustration 17: New code.....	13
Illustration 18: Just created docx file.....	14
Illustration 19: Docx with fresh new content.....	16

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 a detailed description can be found in 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](#)

With this steps, we will ready to start developing.

In next paragraphs we will explain in detail how to create three console applications. 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. Convert pdf to docx file

Create a new folder in your Linux machine with the name *Convert_pdf_to_docx*.

Open VS Code and click on the *Explorer* icon at top left corner. Then click in *Open folder*.

From the dialog, open the folder you created previously:

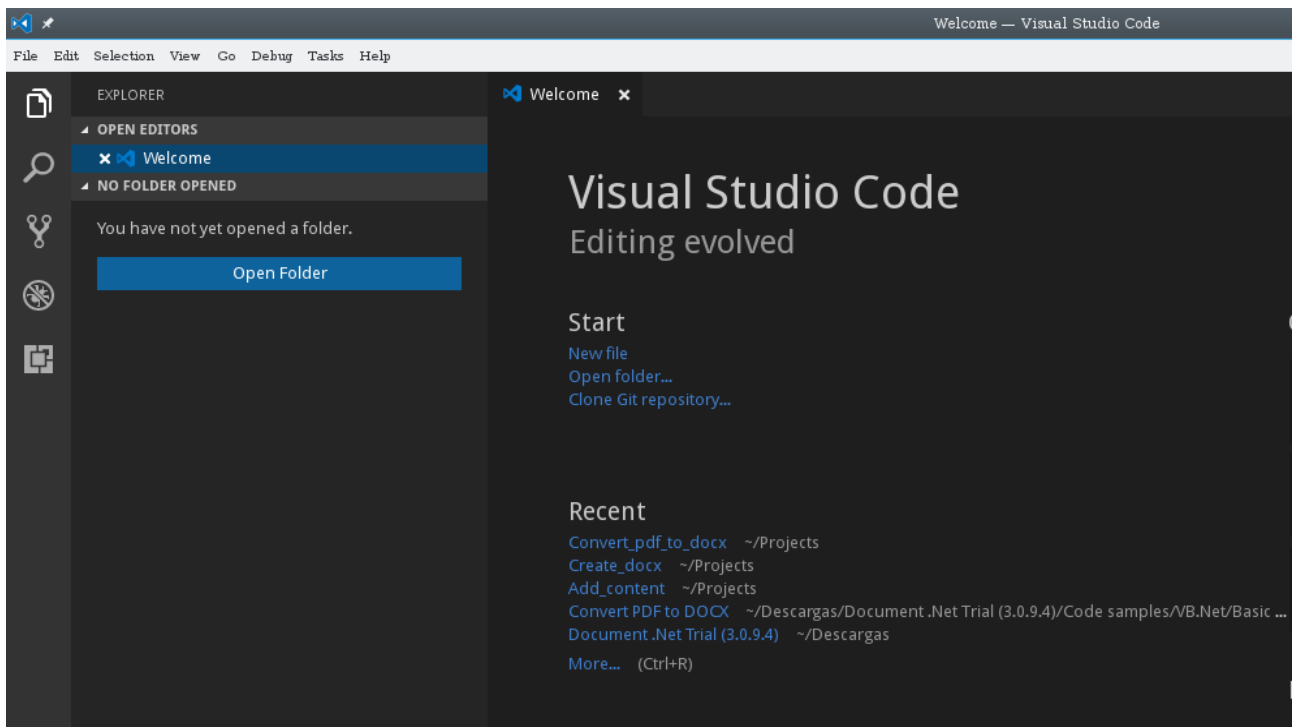


Illustration 1: Explorer

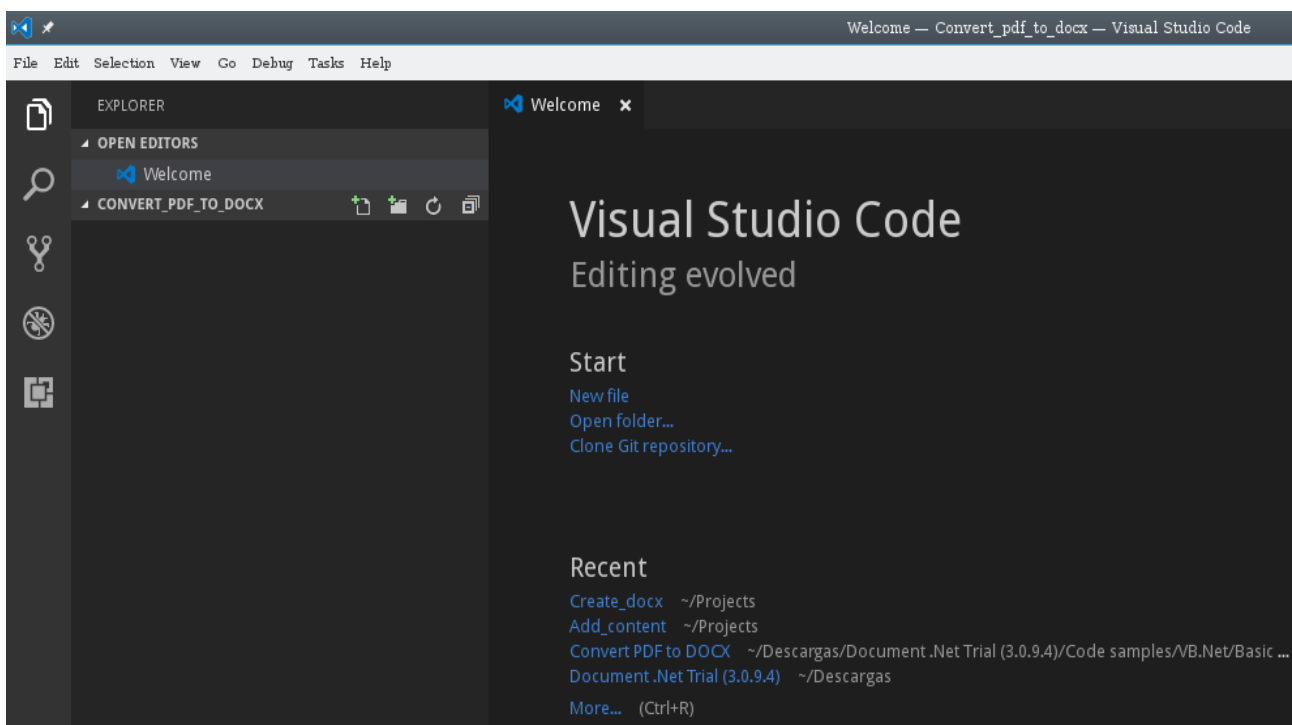


Illustration 2: Showing folder *Convert_pdf_to_docx*

Now, open the integrated console clicking on Ctrl+` :

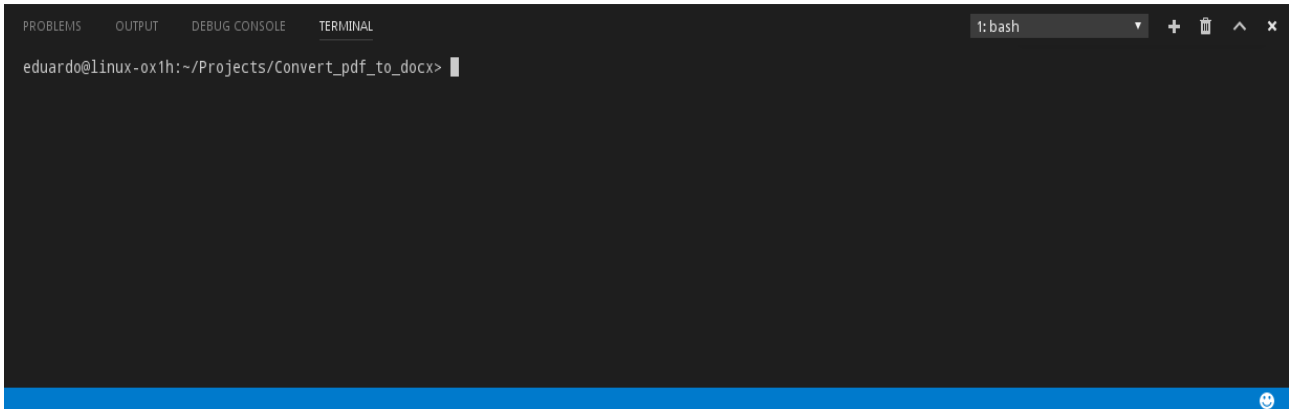


Illustration 3: Integrated console

Now we will create a new console application, using dotnet command.

Type this in console: **dotnet new console**

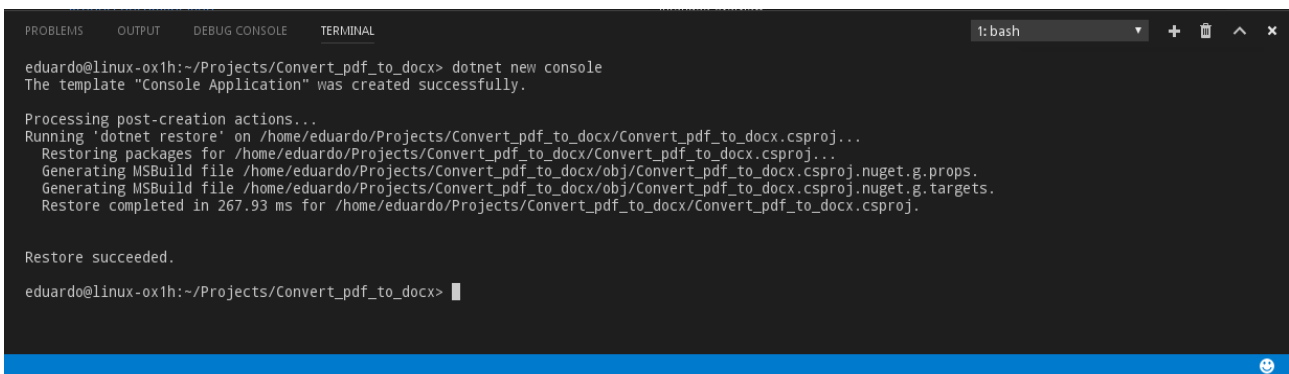


Illustration 4: Creating a new console project

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

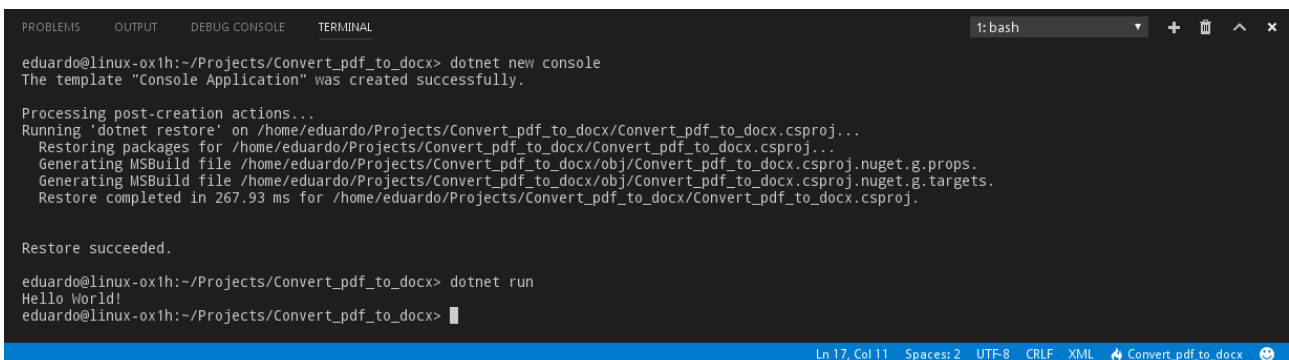


Illustration 5: Running project

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

Now we are going to convert this simple application something more interesting. We will build an application that will convert a pdf file to a docx file.

First of all, we need to add the required dependencies to our project. First of all, we need the package **sautinsoft.document**. In **Nuget** repositories the version we can found is 2.5.3.3 and we need the new beta version that it is not available yet.

In order to add it to our project, we can add it as a local file.

Open file **Convert_pdf_to_docx.csproj** within VS Code to edit it:

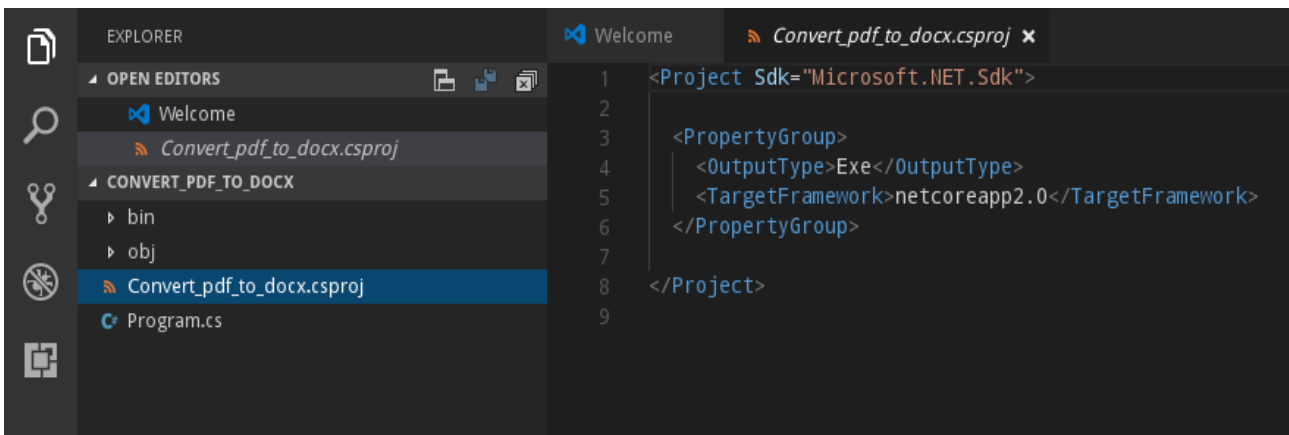


Illustration 6: Editing project configuration file

Add this lines to the file:

```
<ItemGroup>
  <Reference Include="SautinSoft.Document">
    <HintPath>{path_to_the_dll_file}SautinSoft.Document.dll</HintPath>
  </Reference>
</ItemGroup>
```

Set the correct value for the path to dll file in your local machine. Mine is this one:

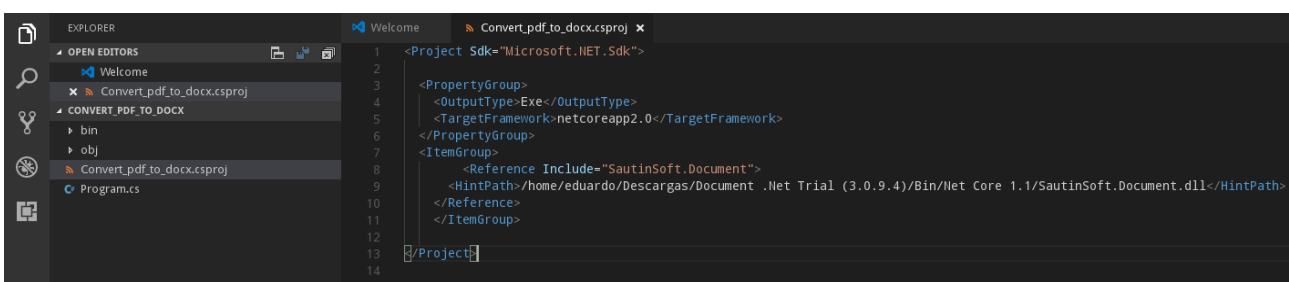


Illustration 7: Adding SaintSoft Document dll reference

We need to include some more items, like the platforms we would build and dependencies of *SautinSoft.Document.dll* library. The final result is this one:

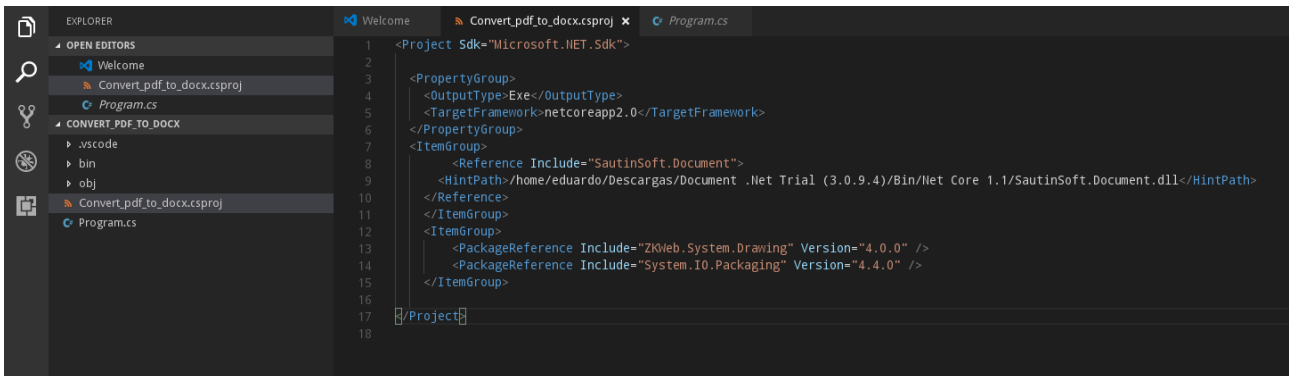


Illustration 8: Adding all needed dependencies

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>
  <PropertyGroup>
    <RuntimeIdentifiers>win10-x64;osx.10.12-x64;opensuse.42.1-x64</RuntimeIdentifiers>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="SautinSoft.Document">
      <HintPath>/home/eduardo/Descargas/Document .Net Trial (3.0.9.4)/Bin/Net Core
      1.1/SautinSoft.Document.dll</HintPath>
    </Reference>
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="ZKWeb.System.Drawing" Version="4.0.0" />
    <PackageReference Include="System.IO.Packaging" Version="4.4.0" />
  </ItemGroup>
</Project>
```

VS Code will warn you about new not resolved dependencies. Click on **Yes** and **Restore** to resolve these dependencies:

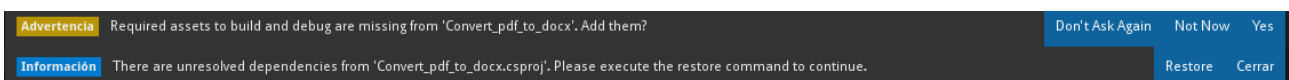


Illustration 9: Adding assets

This is the output in the console as result of restoring command:

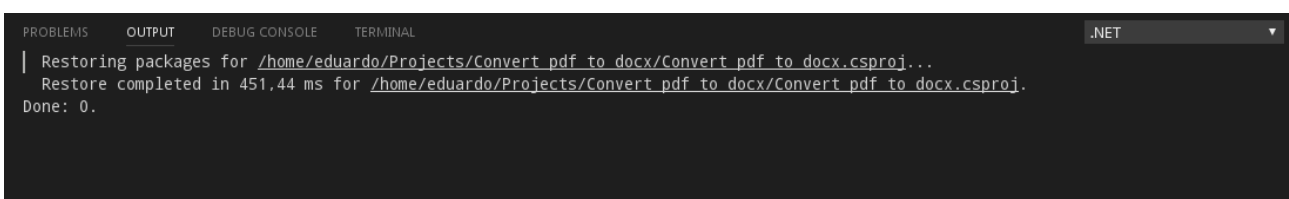


Illustration 10: Restore output

Next step is to type our code to complete the program that will convert a pdf file to docx.

Open file Program.cs within VS Code clicking in its name:

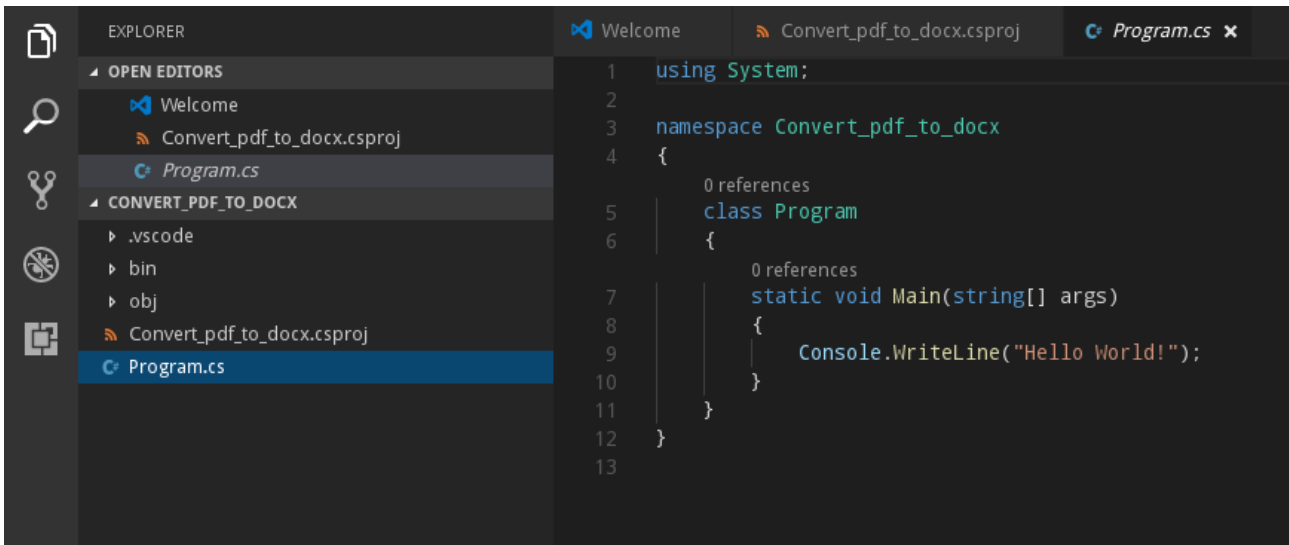


Illustration 11: Editing Program.cs

Type these lines in it:

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

namespace Convert_pdf_to_docx
{
    class Program
    {
        static int Main(string[] args)
        {
            if (args.Length < 2)
            {
                System.Console.WriteLine("Please enter two file names. The first input pdf file and the second output file name");
                return 1;
            }
            return ConvertPdfToDocx(args);
        }
    }
}
/// <summary>
```

```
/// Load an existing PDF and save it as new DOCX document.
/// </summary>
public static int ConvertPdfToDocx(string[] files)
{
    string pdfFile = Path.GetFullPath(files[0]);
    string docxFile = Path.GetFullPath(files[1]);

    //DocumentCore.Serial = "put your serial here";
    int status = 0;
    try
    {
        PdfLoadOptions pdfOptions = new PdfLoadOptions();
        pdfOptions.ConversionMode = PdfConversionMode.Flowing;
        pdfOptions.DetectTables = true;
        pdfOptions.RasterizeVectorGraphics = true;
        pdfOptions.FromPage = 1;
        pdfOptions.KeepCharScaleAndSpacing = true;

        DocumentCore pdf = DocumentCore.Load(pdfFile, pdfOptions);

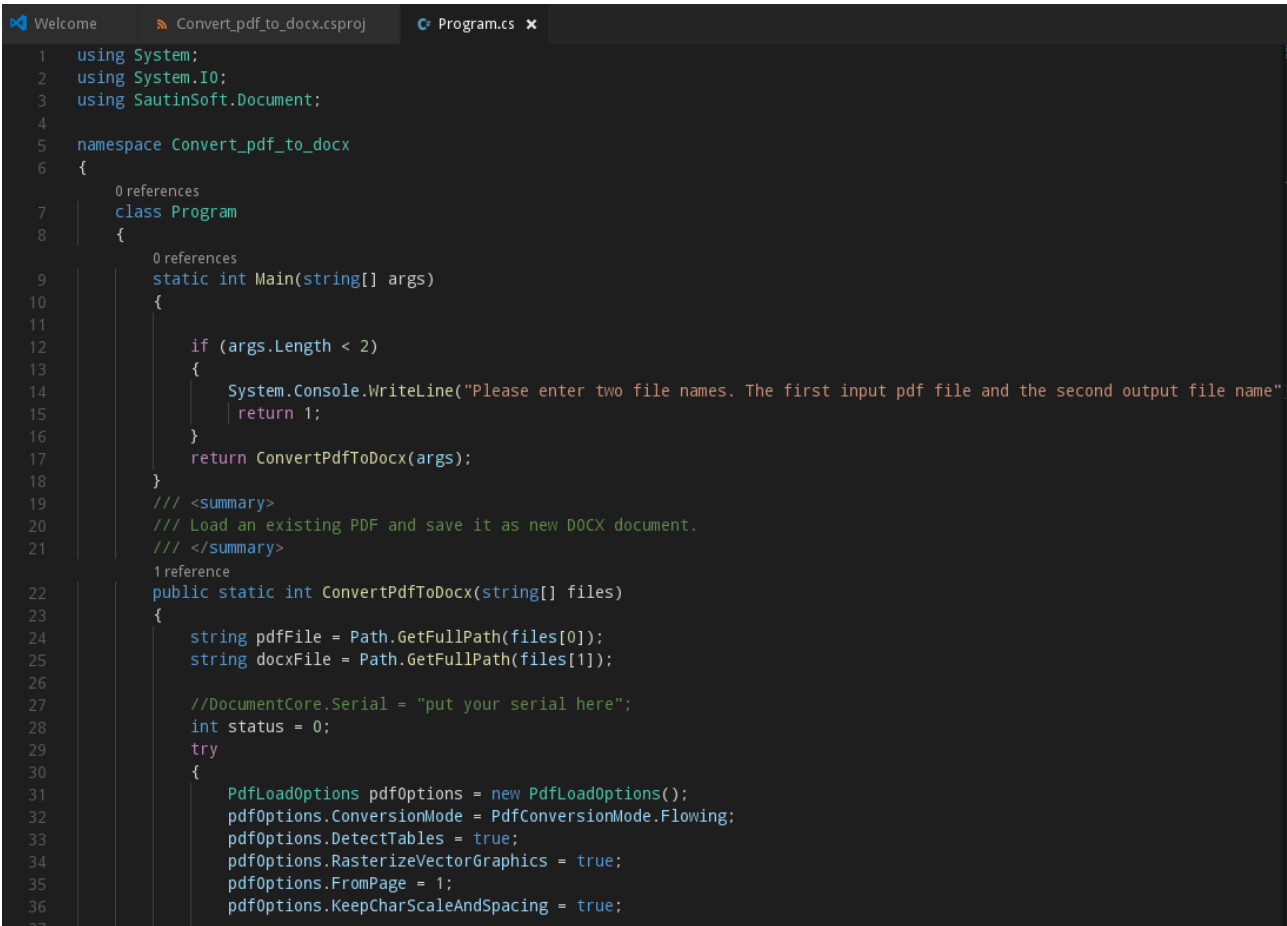
        pdf.Save(docxFile);
    } catch(Exception e)
    {
        System.Console.WriteLine(e.Message);
        status = 1;
    }
    // Open resulting DOCX.
    // This code has a bug in current .NET core. Using alternative below
    //System.Diagnostics.Process.Start(docxFile);
    System.Diagnostics.Process process = new System.Diagnostics.Process();
    System.Diagnostics.ProcessStartInfo startInfo = new System.Diagnostics.ProcessStartInfo();
    startInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
    startInfo.FileName = "soffice";
    startInfo.Arguments = docxFile;
    process.StartInfo = startInfo;
```



```
    process.Start();

    return status;
}
}
}
```

Our code within VS Code editor:



The screenshot shows the Visual Studio Code editor with a C# file named Program.cs. The code is as follows:

```
1 using System;
2 using System.IO;
3 using SautinSoft.Document;
4
5 namespace Convert_pdf_to_docx
6 {
7     0 references
8     class Program
9     {
10         0 references
11         static int Main(string[] args)
12         {
13             if (args.Length < 2)
14             {
15                 System.Console.WriteLine("Please enter two file names. The first input pdf file and the second output file name");
16                 return 1;
17             }
18             return ConvertPdfToDocx(args);
19         }
20         /// <summary>
21         /// Load an existing PDF and save it as new DOCX document.
22         /// </summary>
23         1 reference
24         public static int ConvertPdfToDocx(string[] files)
25         {
26             string pdfFile = Path.GetFullPath(files[0]);
27             string docxFile = Path.GetFullPath(files[1]);
28
29             //DocumentCore.Serial = "put your serial here";
30             int status = 0;
31             try
32             {
33                 PdfLoadOptions pdfOptions = new PdfLoadOptions();
34                 pdfOptions.ConversionMode = PdfConversionMode.Flowing;
35                 pdfOptions.DetectTables = true;
36                 pdfOptions.RasterizeVectorGraphics = true;
37                 pdfOptions.FromPage = 1;
38                 pdfOptions.KeepCharScaleAndSpacing = true;
```

Illustration 12: Inserting new code

Reached this point, we can build our program. Copy a pdf file you want to convert to **Convert_pdf_to_docx** folder:

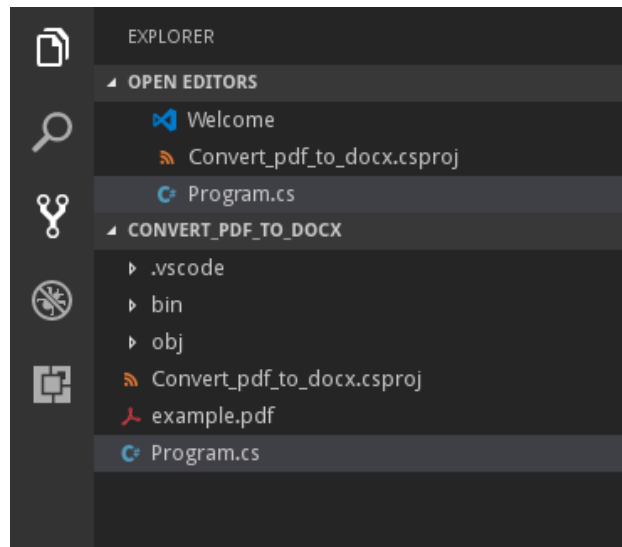


Illustration 13: Add pdf file to convert

Now, type this in console: **dotnet run example.pdf example.docx**

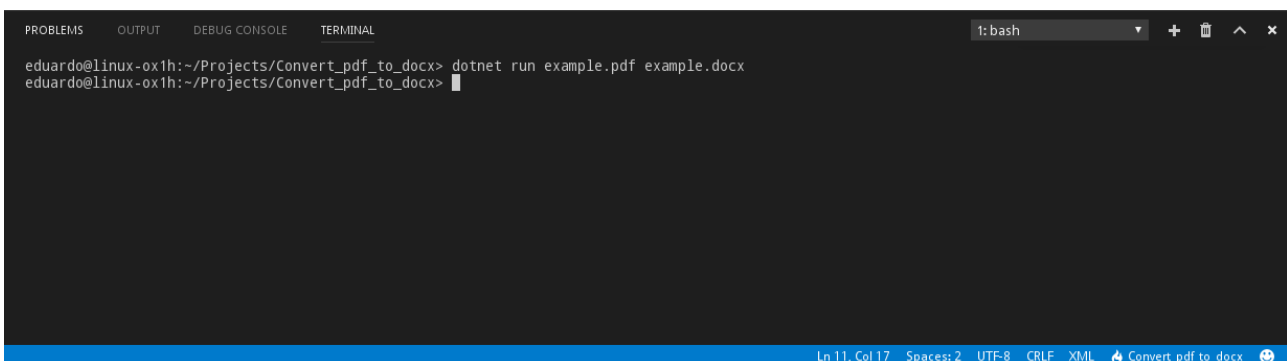


Illustration 14: Run command

A new file **example.docx** is created and it will be opened with LibreOffice writer editor, in the case you have it installed.

Note: It is possible that the native library **libgdiplus.so** cannot be found in your Linux machine. If the pdf document includes images and these are missing in the docx file, create a link to native library libgdiplus.so with this command:

```
sudo ln -s /usr/lib64/libgdiplus.so /usr/lib64/gdiplus
```

Ensure **libgdiplus** is installed in your system. May be that in your machine, prefix to your native libraries would be different.

That's all. We have now a *dll* file that can be executed in your Linux machine using **dotnet** tool.

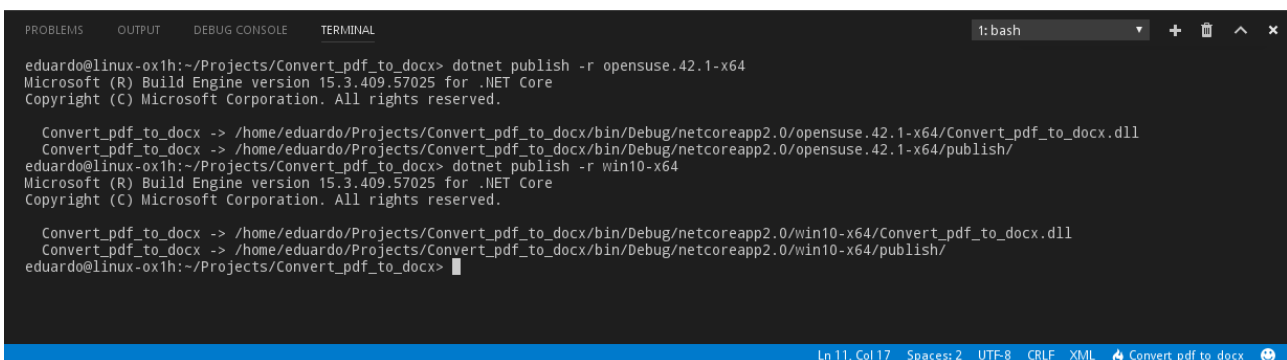
Well, in fact, that's not all. We can “publish” our program to be executed it in any other machine without .Net Core SDK installed. This includes a new native executable file and all its required dependencies.

To do it, you only need to type this in console, in the case you want to publish to Linux OpenSuse Leap:

```
dotnet publish -r opensuse.42.1-x64
```

Or this one to publish for Windows 10:

```
dotnet publish -r win10-x64
```



```
eduardo@linux-ox1h:~/Projects/Convert_pdf_to_docx> dotnet publish -r opensuse.42.1-x64
Microsoft (R) Build Engine version 15.3.409.57025 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

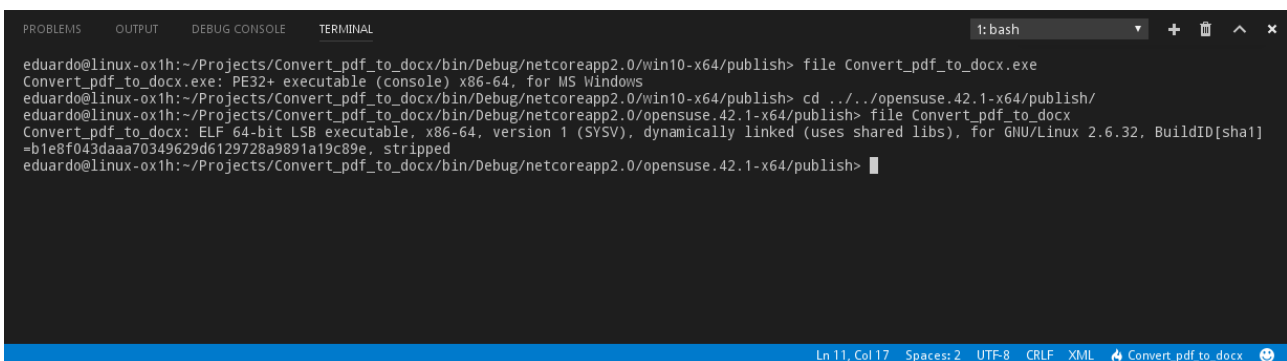
    Convert_pdf_to_docx -> /home/eduardo/Projects/Convert_pdf_to_docx/bin/Debug/netcoreapp2.0/opensuse.42.1-x64/Convert_pdf_to_docx.dll
    Convert_pdf_to_docx -> /home/eduardo/Projects/Convert_pdf_to_docx/bin/Debug/netcoreapp2.0/opensuse.42.1-x64/publish/
eduardo@linux-ox1h:~/Projects/Convert_pdf_to_docx> dotnet publish -r win10-x64
Microsoft (R) Build Engine version 15.3.409.57025 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

    Convert_pdf_to_docx -> /home/eduardo/Projects/Convert_pdf_to_docx/bin/Debug/netcoreapp2.0/win10-x64/Convert_pdf_to_docx.dll
    Convert_pdf_to_docx -> /home/eduardo/Projects/Convert_pdf_to_docx/bin/Debug/netcoreapp2.0/win10-x64/publish/
eduardo@linux-ox1h:~/Projects/Convert_pdf_to_docx>
```

Illustration 15: Publish program

Congratulations! You already have your fresh new program that can be executed in virtually any modern SO!

The only you need to do, is to copy the content of the publish folder in any supported machine and to execute the native executable:



```
eduardo@linux-ox1h:~/Projects/Convert_pdf_to_docx/bin/Debug/netcoreapp2.0/win10-x64/publish> file Convert_pdf_to_docx.exe
Convert_pdf_to_docx.exe: PE32+ executable (console) x86-64, for MS Windows
eduardo@linux-ox1h:~/Projects/Convert_pdf_to_docx/bin/Debug/netcoreapp2.0/win10-x64/publish> cd ../../opensuse.42.1-x64/publish/
eduardo@linux-ox1h:~/Projects/Convert_pdf_to_docx/bin/Debug/netcoreapp2.0/opensuse.42.1-x64/publish> file Convert_pdf_to_docx
Convert_pdf_to_docx: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=b1e8f043daaa70349629d6129728a9891a19c89e, stripped
eduardo@linux-ox1h:~/Projects/Convert_pdf_to_docx/bin/Debug/netcoreapp2.0/opensuse.42.1-x64/publish>
```

Illustration 16: Native files

3. Create docx from scratch

Now we will going to develop a program that will be able to create a new docx document and to add some content in it.

As we did before, create a new folder and name it Create_docx. Open this folder within VS Code and repeat the same steps as done before, creating a new console project, adding dependencies and so on.

One you have done and are ready to code your new program, type this within Program.cs file:

```
{
using System;
using System.IO;
using SautinSoft.Document;

namespace Create_docx
{
    class Program
    {
        static void Main(string[] args)
        {

            // Let's create a simple DOCX document.
            DocumentCore docx = new DocumentCore();

            // Add new section.
            Section section = new Section(docx);
            docx.Sections.Add(section);

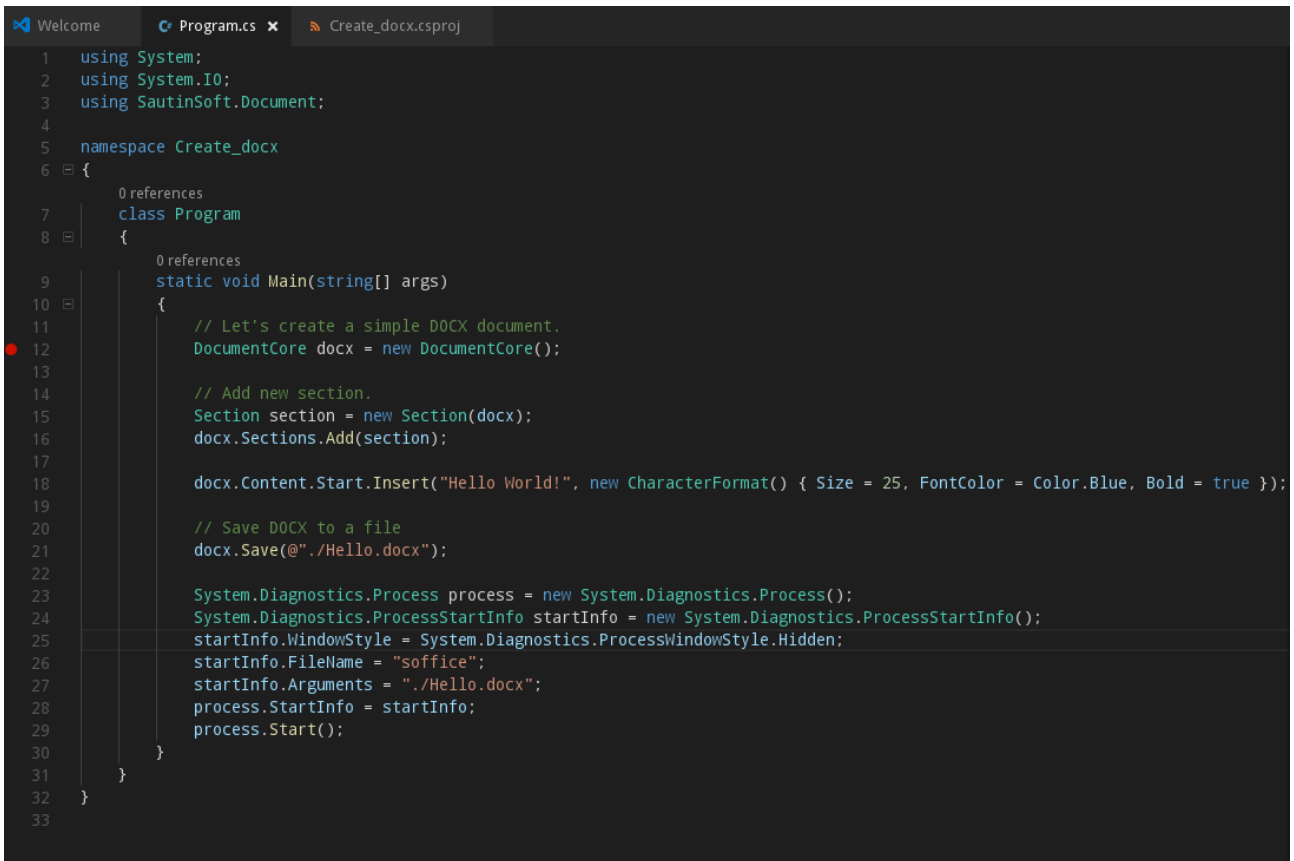
            docx.Content.Start.Insert("Hello World!", new CharacterFormat() { Size = 25, FontColor =
            Color.Blue, Bold = true });

            // Save DOCX to a file
            docx.Save(@"./Hello.docx");

            System.Diagnostics.Process process = new System.Diagnostics.Process();
            System.Diagnostics.ProcessStartInfo startInfo = new System.Diagnostics.ProcessStartInfo();
            startInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
            startInfo.FileName = "soffice";
            startInfo.Arguments = "./Hello.docx";
            process.StartInfo = startInfo;
            process.Start();

        }
    }
}
```

This is how it is shown within VS Code editor:



```
1 using System;
2 using System.IO;
3 using SautinSoft.Document;
4
5 namespace Create_docx
6 {
7     0 references
8     class Program
9     {
10        0 references
11        static void Main(string[] args)
12        {
13            // Let's create a simple DOCX document.
14            DocumentCore docx = new DocumentCore();
15
16            // Add new section.
17            Section section = new Section(docx);
18            docx.Sections.Add(section);
19
20            docx.Content.Start.Insert("Hello World!", new CharacterFormat() { Size = 25, FontColor = Color.Blue, Bold = true });
21
22            // Save DOCX to a file
23            docx.Save(@"./Hello.docx");
24
25            System.Diagnostics.Process process = new System.Diagnostics.Process();
26            System.Diagnostics.ProcessStartInfo startInfo = new System.Diagnostics.ProcessStartInfo();
27            startInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
28            startInfo.FileName = "soffice";
29            startInfo.Arguments = "./Hello.docx";
30            process.StartInfo = startInfo;
31            process.Start();
32        }
33    }
```

Illustration 17: New code

Remember to save your work and run your new program: **dotnet run**

LibreOffice writer will open and displays your new docx file:

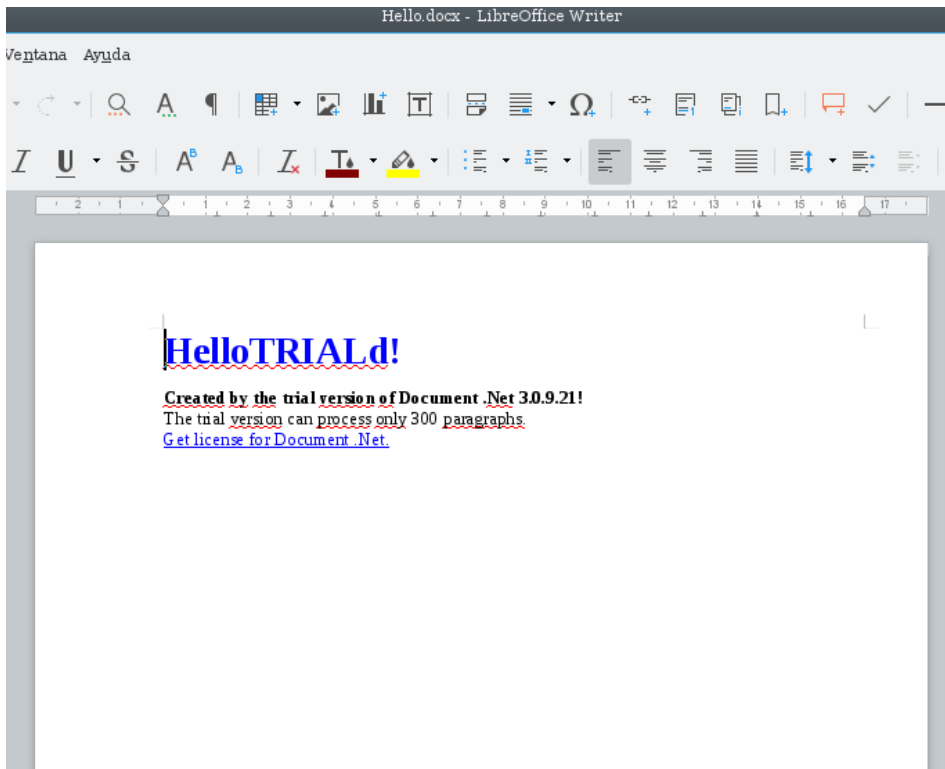


Illustration 18: Just created docx file

If you want to execute this new program in a machine without installing .Net SDK, remember you can publish the project to obtain a native program and all dependencies required.

4. Add content to docx file

Perform the same steps as you did before, but now create a folder named **Add_content**

Type these lines in **Program.cs** file:

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

namespace Add_content
{
    class Program
    {
        static void Main(string[] args)
        {
            AddContent();
        }
        /// <summary>
        /// Create a new document and add some content inside it.
        /// </summary>
    }
}
```

```

public static void AddContent()
{
    // Set a path to our docx file.
    string docxPath = @"/Hello.docx";

    // Let's create a simple DOCX document.
    DocumentCore docx = new DocumentCore();
    //DocumentCore.Serial = "put your serial here";

    // Add new section.
    Section section = new Section(docx);
    docx.Sections.Add(section);

    // Let's set page size A4.
    section.PageSetup.PaperType = PaperType.A4;

    // Add two paragraphs using different ways:

    // Way 1: Add 1st paragraph.
    section.Blocks.Add(new Paragraph(docx, "This is a first line in 1st paragraph!"));
    Paragraph par1 = section.Blocks[0] as Paragraph;
    par1.ParagraphFormat.Alignment = HorizontalAlignment.Center;

    // Let's add a second line.
    par1.Inlines.Add(new SpecialCharacter(docx, SpecialCharacterType.LineBreak));
    par1.Inlines.Add(new Run(docx, "Let's type a second line."));

    // Let's change font name, size and color.
    CharacterFormat cf = new CharacterFormat() { FontName = "Verdana", Size = 16,
    FontColor = Color.Orange };
    foreach (Inline inline in par1.Inlines)
        if (inline is Run)
            (inline as Run).CharacterFormat = cf.Clone();

    // Way 2 (easy): Add 2nd paragraph using another way.
    docx.Content.End.Insert("\nThis is a first line in 2nd paragraph.", new CharacterFormat()
    { Size = 25, FontColor = Color.Blue, Bold = true });
    SpecialCharacter lBr = new SpecialCharacter(docx, SpecialCharacterType.LineBreak);
    docx.Content.End.Insert(lBr.Content);
    docx.Content.End.Insert("This is a second line.", new CharacterFormat() { Size = 20,
    FontColor = Color.DarkGreen, UnderlineStyle = UnderlineType.Single });

    // Save DOCX to a file
    docx.Save(docxPath);

    // Open the result for demonstation purposes.
    // NET Core bug here
    //System.Diagnostics.Process.Start(docxPath);

    // Alternative way
    System.Diagnostics.Process process = new System.Diagnostics.Process();
    System.Diagnostics.ProcessStartInfo startInfo = new System.Diagnostics.ProcessStartInfo();
    startInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
    startInfo.FileName = "soffice";
    startInfo.Arguments = "/Hello.docx";
    process.StartInfo = startInfo;
    process.Start();
}
}
}

```

Same as before, type **dotnet run** to execute the program.

The just created docx file will be opened:

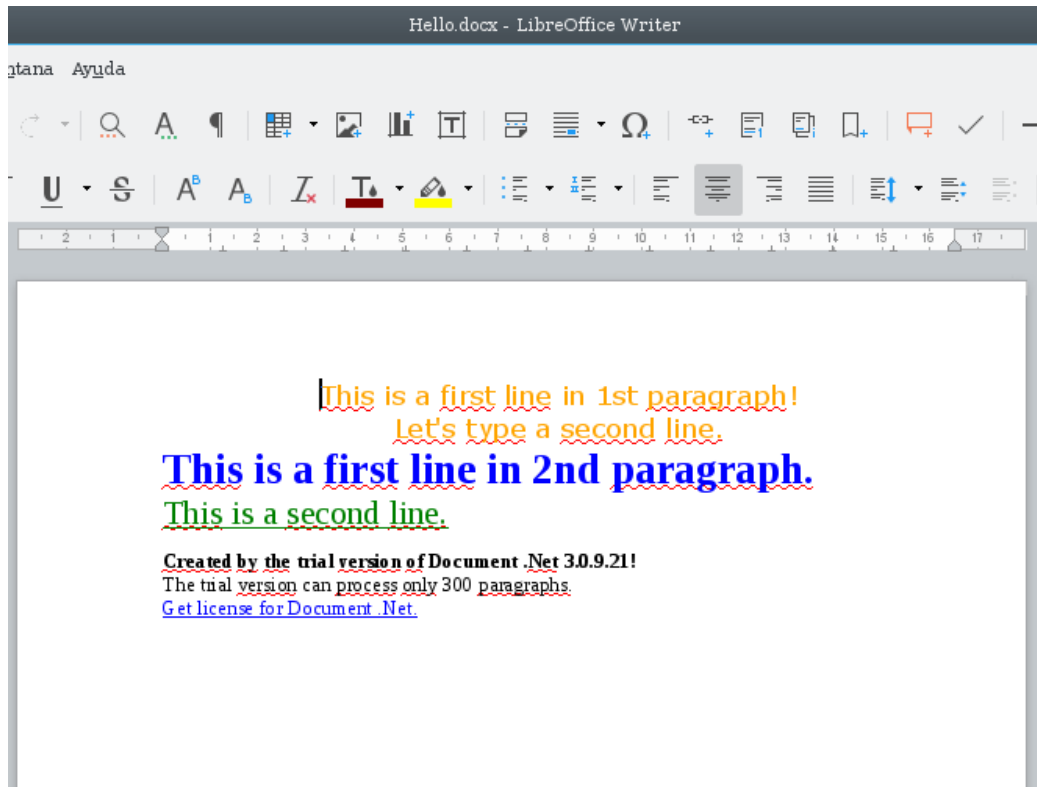


Illustration 19: Docx with fresh new content