Document .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.

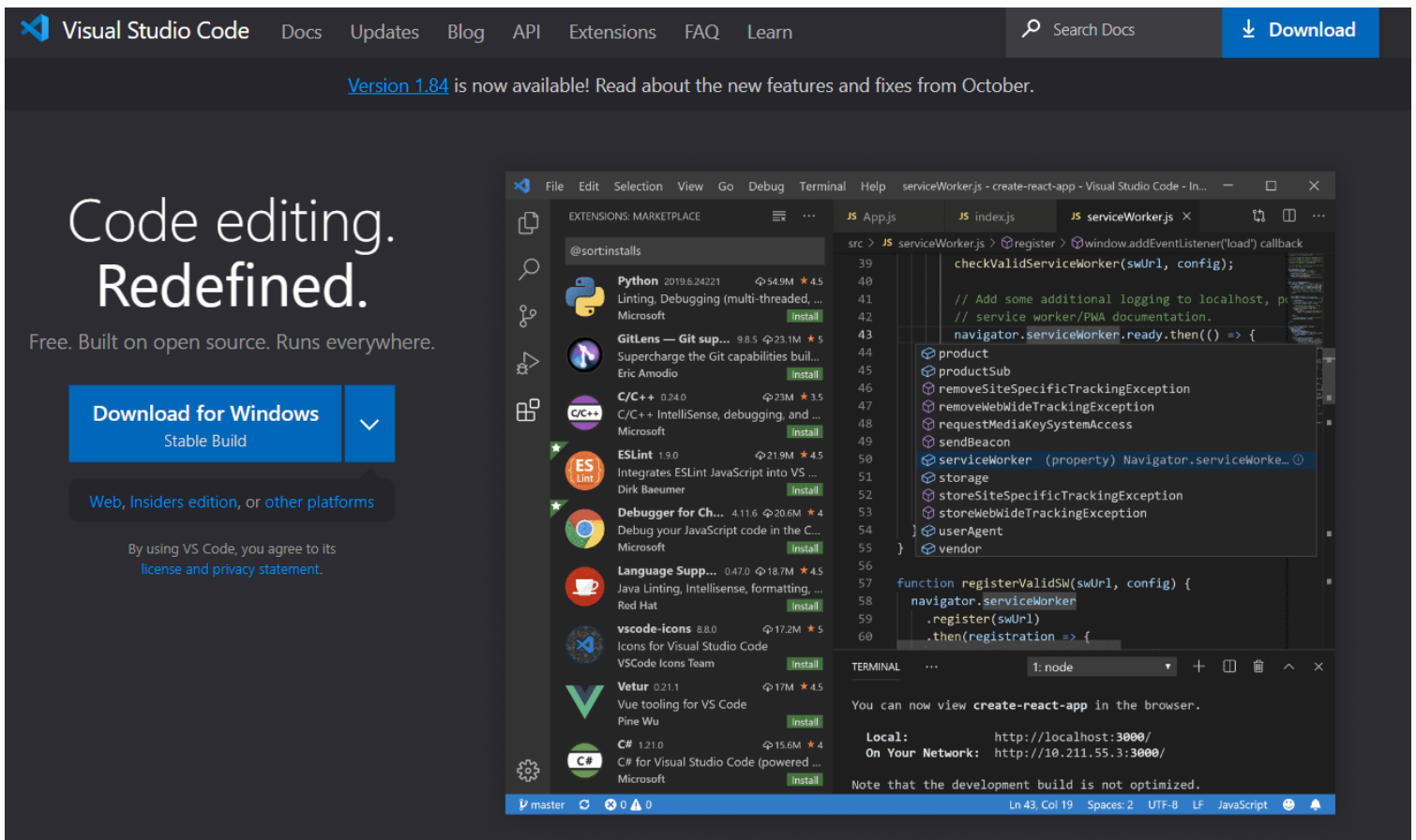Both installations are very easy and the detailed description can be found by these two links:

Install .NET Core SDK for Linux.
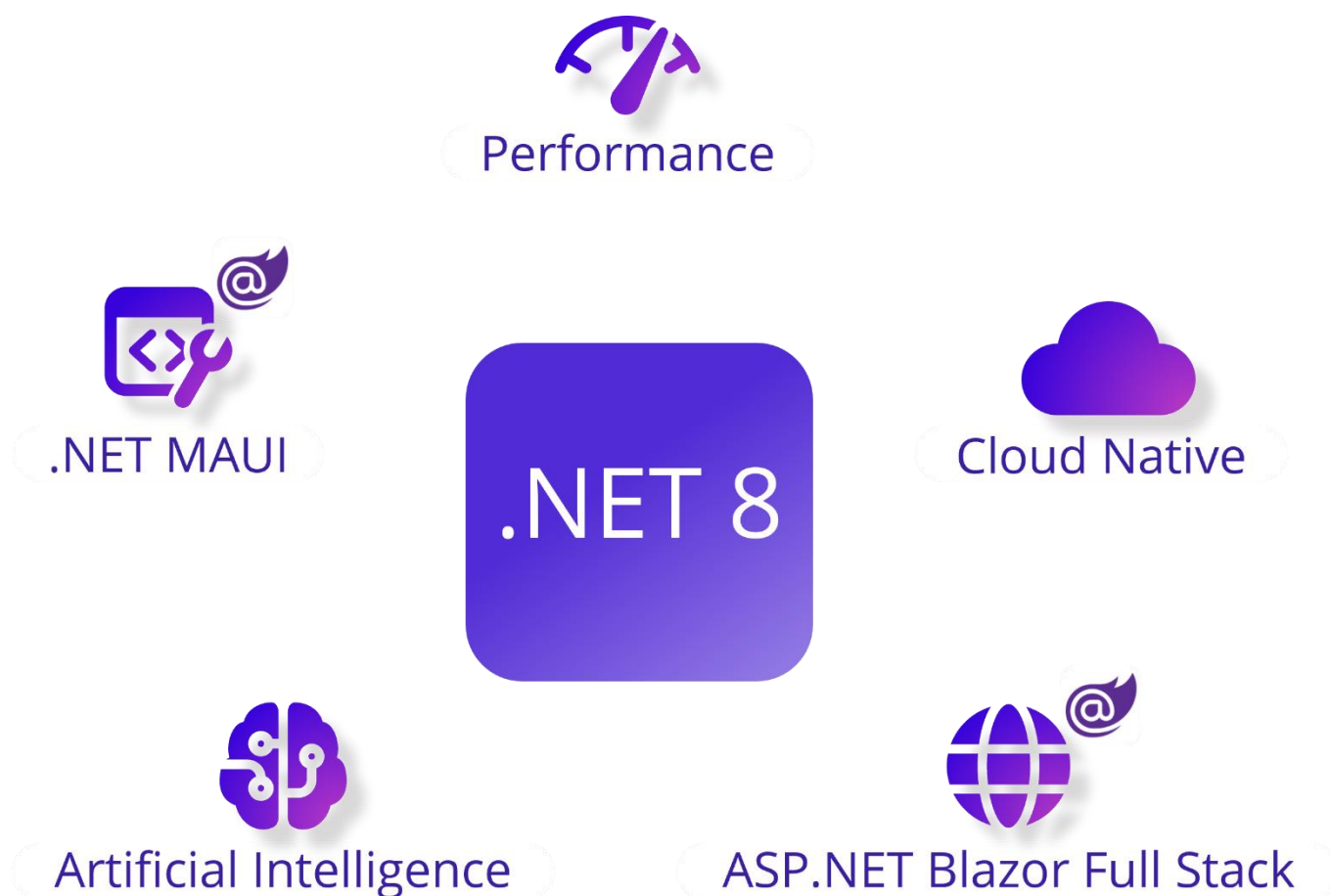
Install VS Code for Linux.

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

Install C# extension.

1.1. Check the installed Fonts availability

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
   ```
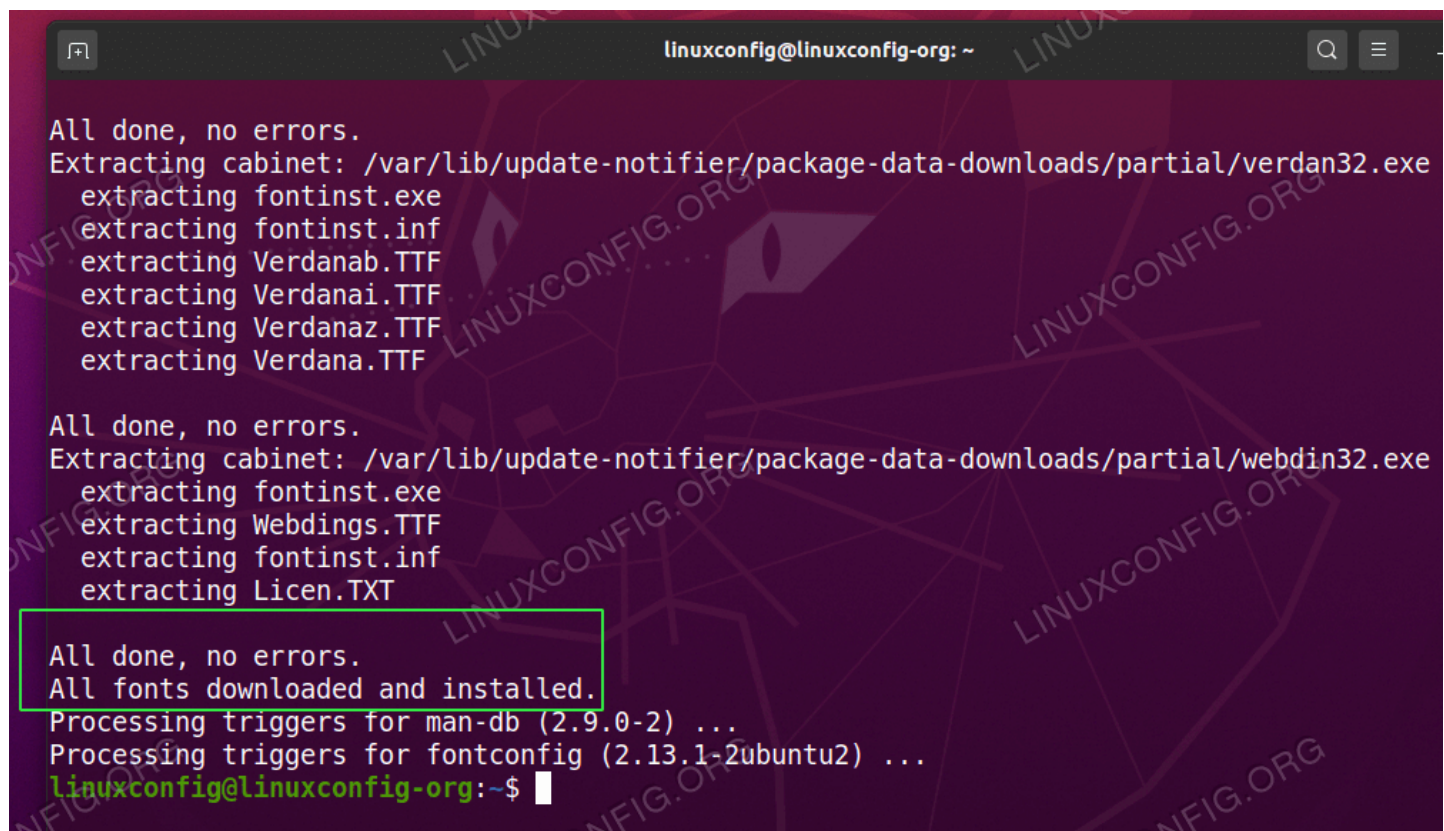
8. Restart X font server

   ```
   # /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
```

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.
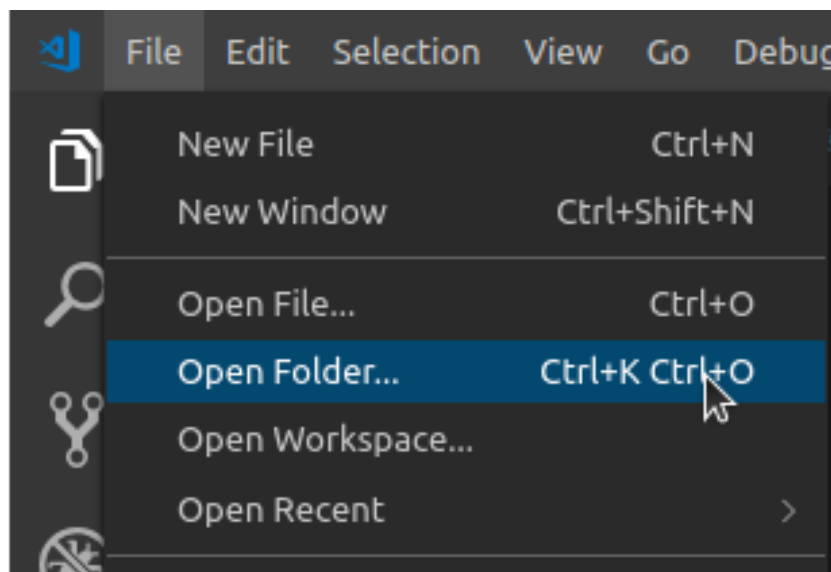
2. Creating "Convert PDF to DOCX" application

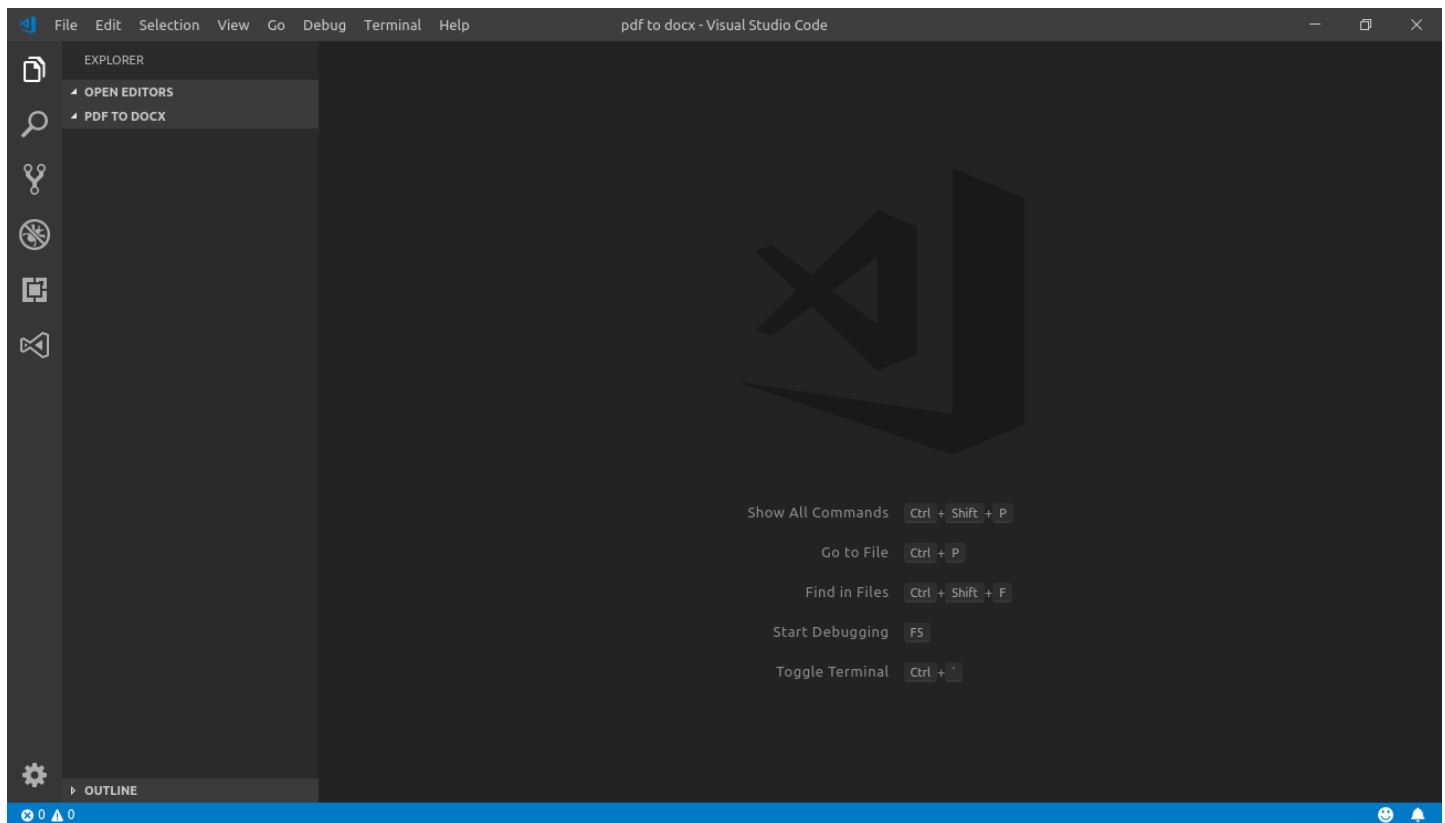Create a new folder in your Linux machine with the name *pdf to docx.*

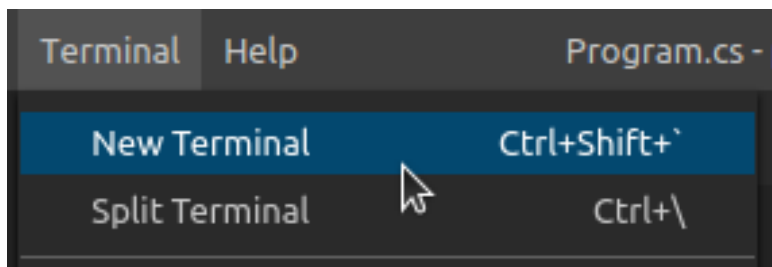For example, let's create the folder "*pdf to docx*" on Desktop (Right click-> New Folder):
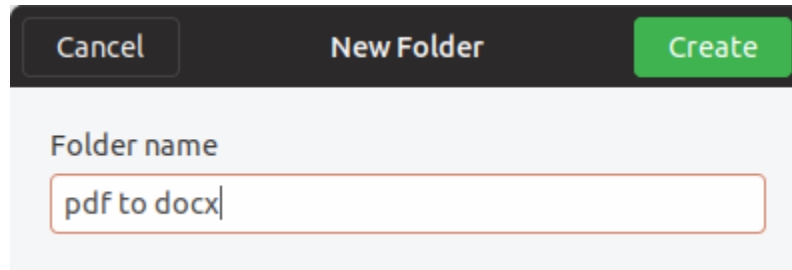
Open VS Code and click in the menu **File->Open Folder**. From the dialog, open the folder you've created previously:
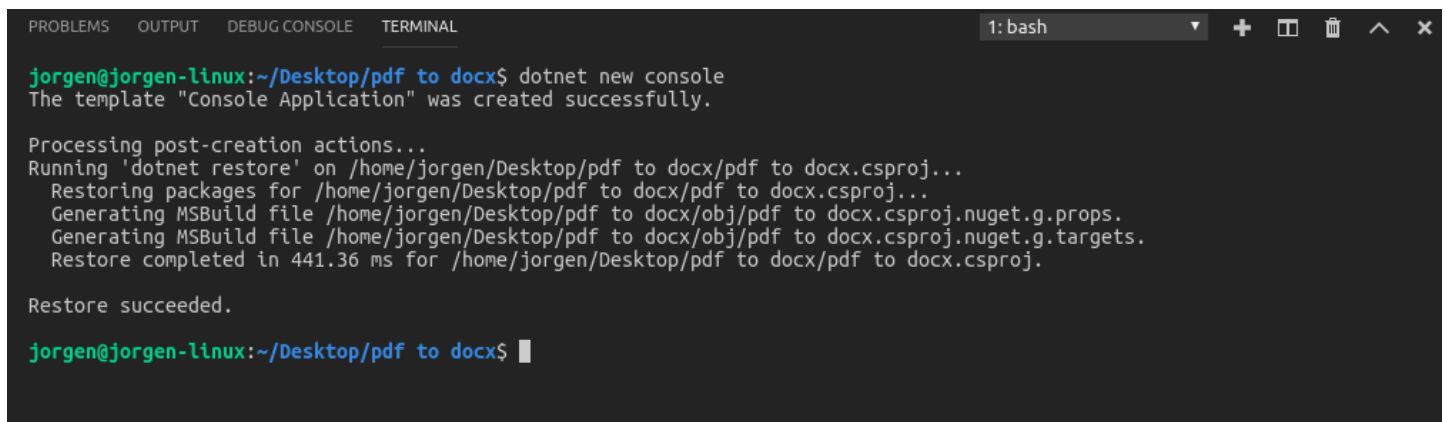
Next you will see the similar screen:

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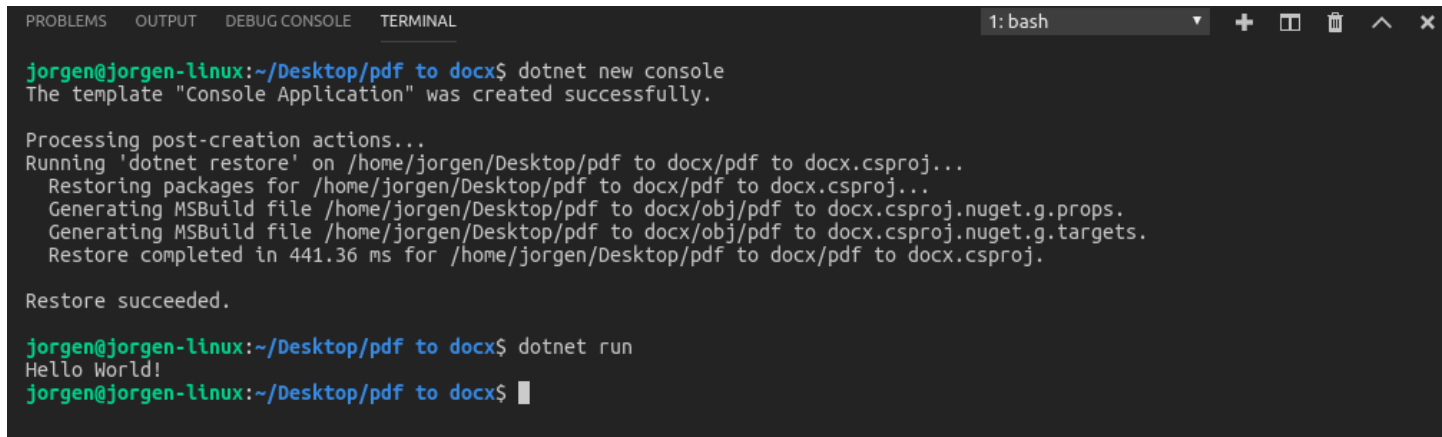

Create a new console application, using **dotnet** command.

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***

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 a pdf file to a docx file.

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

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

Add these lines into the file "**_pdf to docx.csproj_**":

<PackageReference Include="System.IO.Packaging" Version="4.5.0" />

<PackageReference Include="System.Text.Encoding.CodePages" Version="4.5.0" />

<PackageReference Include="Pkcs11Interop" Version="5.1.2" />

<PackageReference Include="Portable.BouncyCastle" Version="1.9.0" />

<PackageReference Include="SkiaSharp" Version="2.88.7" />

<PackageReference Include="SkiaSharp.HarfBuzz" Version="2.88.7" />

<PackageReference Include="Svg.Skia" Version="1.0.0.18" />

<PackageReference Include="System.Resources.Extensions" Version="6.0.0" />
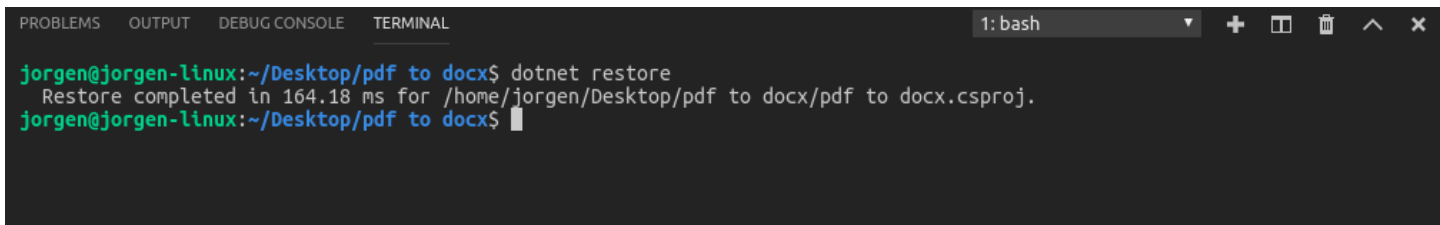
It's the reference to **_sautinsoft.document_** package from Nuget.

At the moment of writing this manual, the latest version of **_sautinsoft.document_** was

2024.X. But you may specify the latest version, to know what is the latest, follow:

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

At once as we've added the package reference, we have to save the "**_pdf to docx.csproj_**"

and restore the added package.

Follow to the **_Terminal_** and type the command: **_dotnet restore_**

Good, now our application has the reference to **_sautinsoft.document_** package and we can

write the code to convert pdf to docx and other formats.

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

The code:

```
using System;
using System.IO;
using SautinSoft.Document;
namespace pdf_to_docx
{
    class Program
    {
        static void Main(string[] args)
        {
            string inpFile = @"/example.pdf";
            string outFile = Path.ChangeExtension(inpFile,".docx");
            DocumentCore dc = DocumentCore.Load(inpFile, new PdfLoadOptions()
            {
                ConversionMode = PdfConversionMode.Flowing,
                PreserveEmbeddedFonts = true
            });
            dc.Save(outFile);
        }
    }
}
```
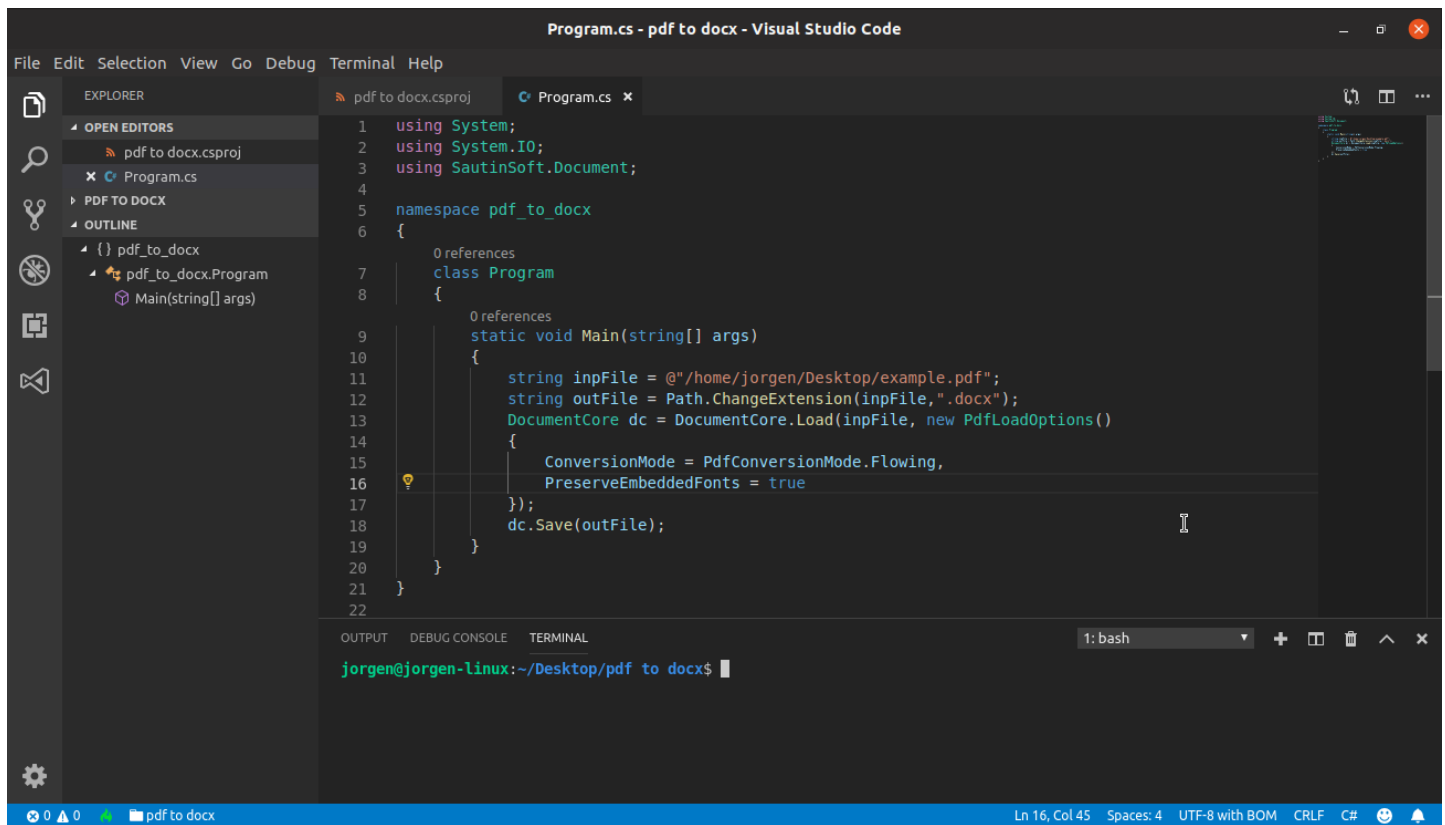
To make tests, we need an input PDF document. For our tests, let's place a PDF file with the name "example.pdf" at the Desktop.

If we open this file in the default PDF Viewer, we'll its contents:
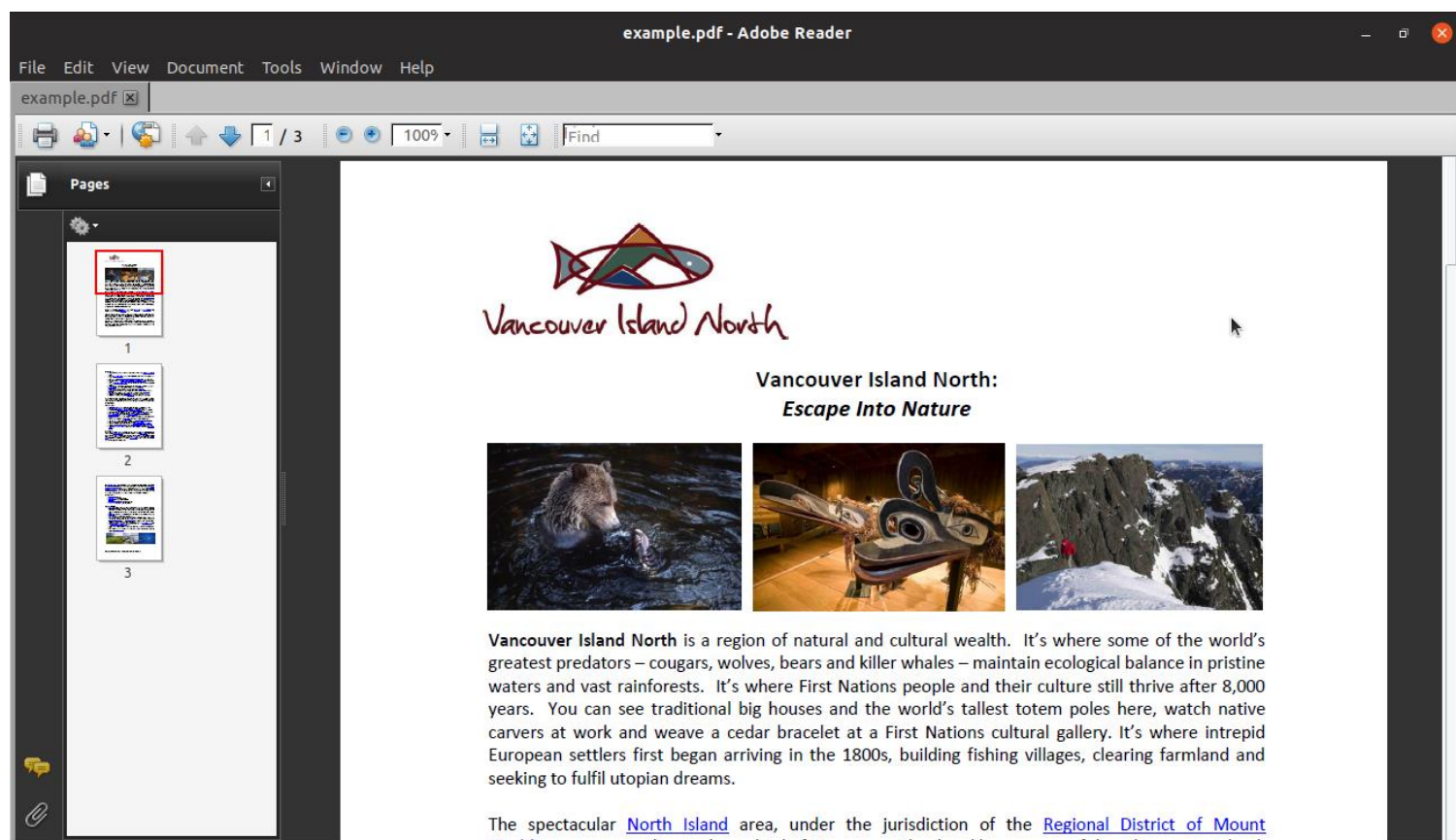
Launch our application and convert the "example.pdf" into "example.docx", type the command: ***dotnet run***
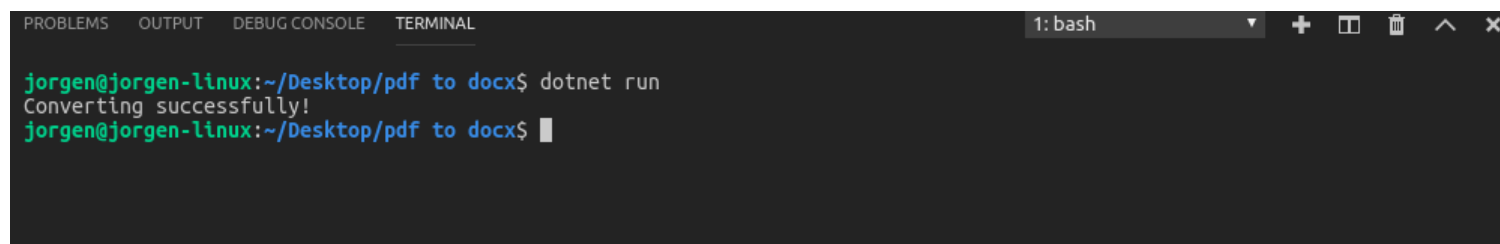
If you don't see any exceptions, everything is fine and we can check the result produced by the Document .Net library.

The new file "example.docx" has to appear on the Desktop:

Open the result in LibreOffice:

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

3. Creating new DOCX document from scratch

Now we're going to develop a new application 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.
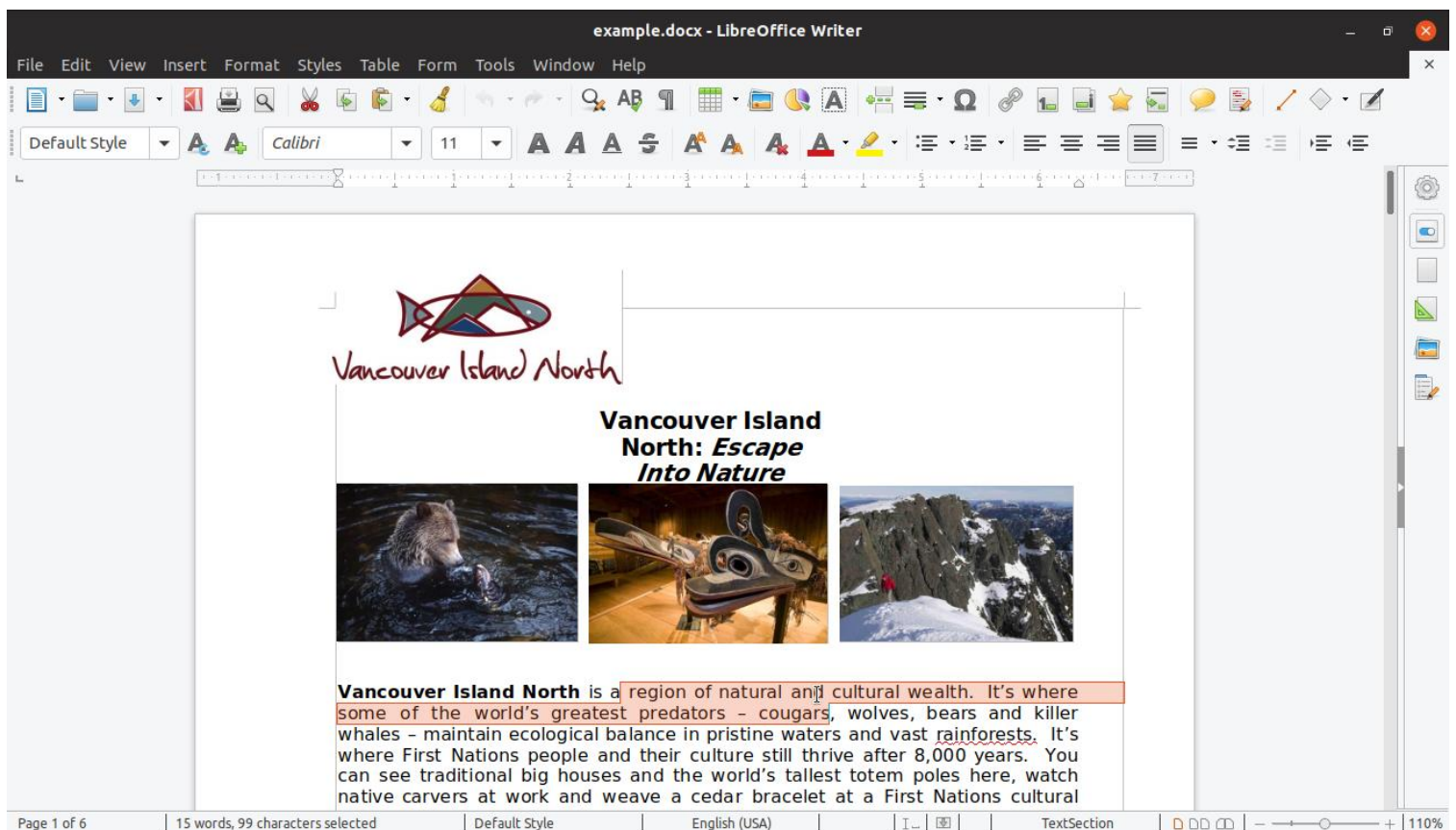
Once you have done and are ready to code your new program, type this within **Program.cs** as shown in the picture below (the complete code is after the picture):
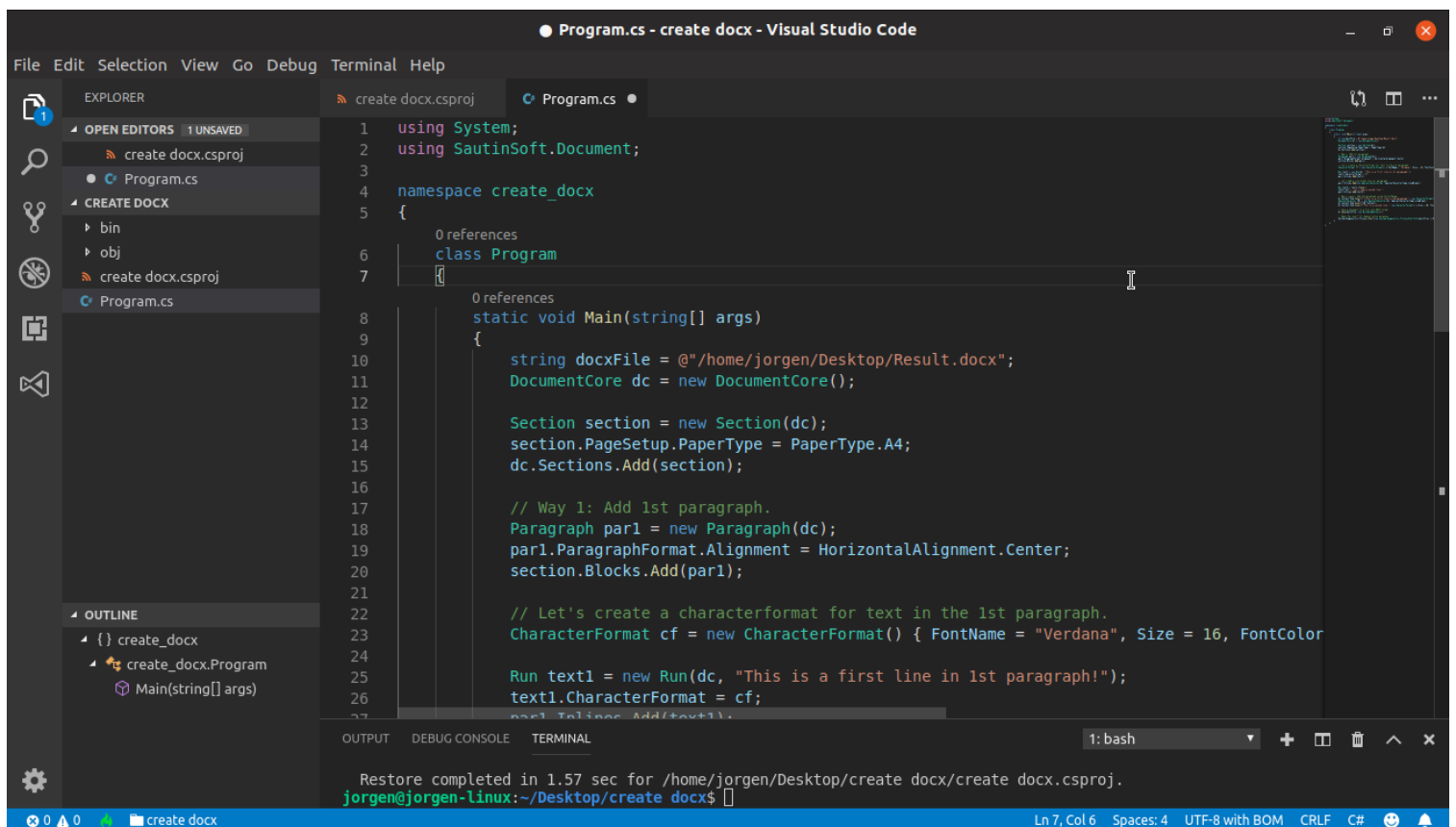
The code:

```
using System;
using SautinSoft.Document;

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

```csharp
{
    string docxFile = @"/home/jorgen/Desktop/Result.docx";
    DocumentCore dc = new DocumentCore();

    Section section = new Section(dc);
    section.PageSetup.PaperType = PaperType.A4;
    dc.Sections.Add(section);

    // Way 1: Add 1st paragraph.
    Paragraph par1 = new Paragraph(dc);
    par1.ParagraphFormat.Alignment = HorizontalAlignment.Center;
    section.Blocks.Add(par1);

    // Let's create a characterformat for text in the 1st paragraph.
    CharacterFormat cf = new CharacterFormat() { FontName = "Verdana", Size = 16,
FontColor = Color.Orange };

    Run text1 = new Run(dc, "This is a first line in 1st paragraph!");
    text1.CharacterFormat = cf;
    par1.Inlines.Add(text1);

    // Let's add a line break into our paragraph.
    par1.Inlines.Add(new SpecialCharacter(dc, SpecialCharacterType.LineBreak));

    Run text2 = text1.Clone();
    text2.Text = "Let's type a second line.";
    par1.Inlines.Add(text2);

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

    // Save a document to a file into DOCX format.
    dc.Save(docxFile, new DocxSaveOptions());

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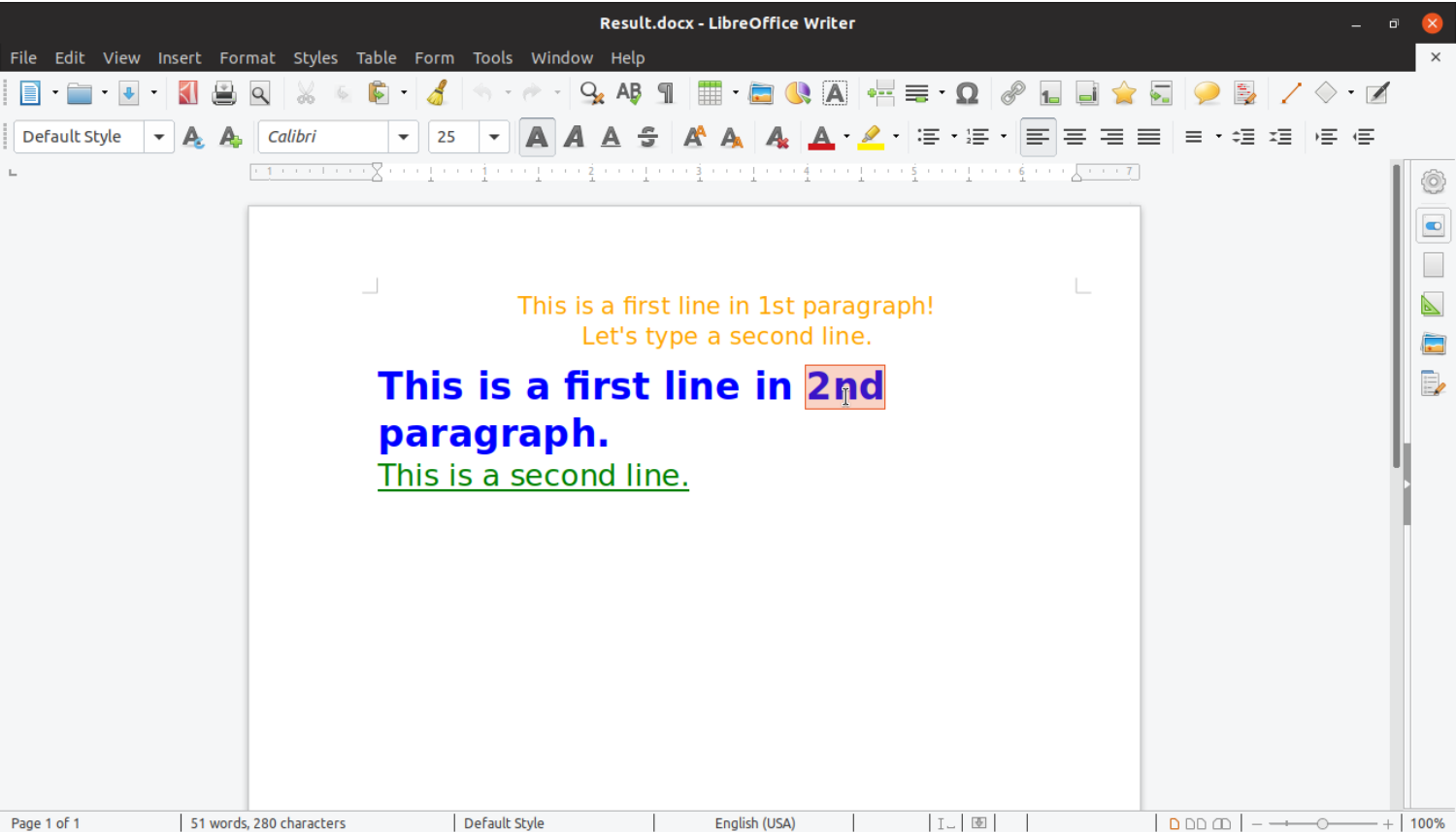
```
        }
    }
```

Launch our application to create a new DOCX document, type the command: ***dotnet run***

If you don't see any exceptions, the produced DOCX file will be opened automatically in the default DOCX viewer (in our case it's LibreOffice):

Well done! You have created the "Create DOCX" 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!