

# Pdf Focus.Net

*(Multi-platform .Net library)*

[SautinSoft](#)

# Linux development manual

## Table of Contents

1. Preparing environment .....	2
1.1. Check the installed Fonts availability.....	3
2. Creating "Convert PDF to DOCX" application.....	5

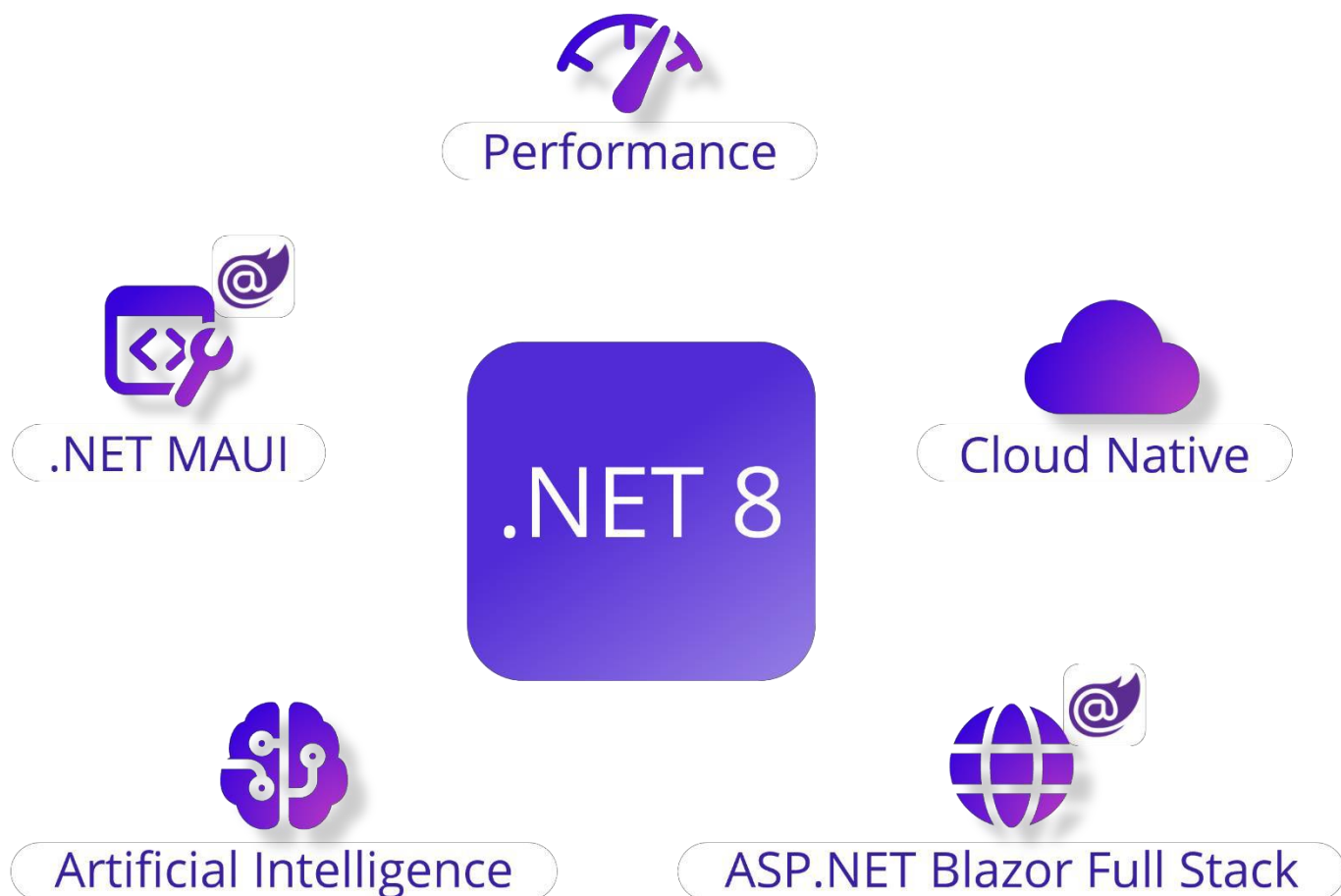
# 1. Preparing environment

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

We need to install .NET 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:

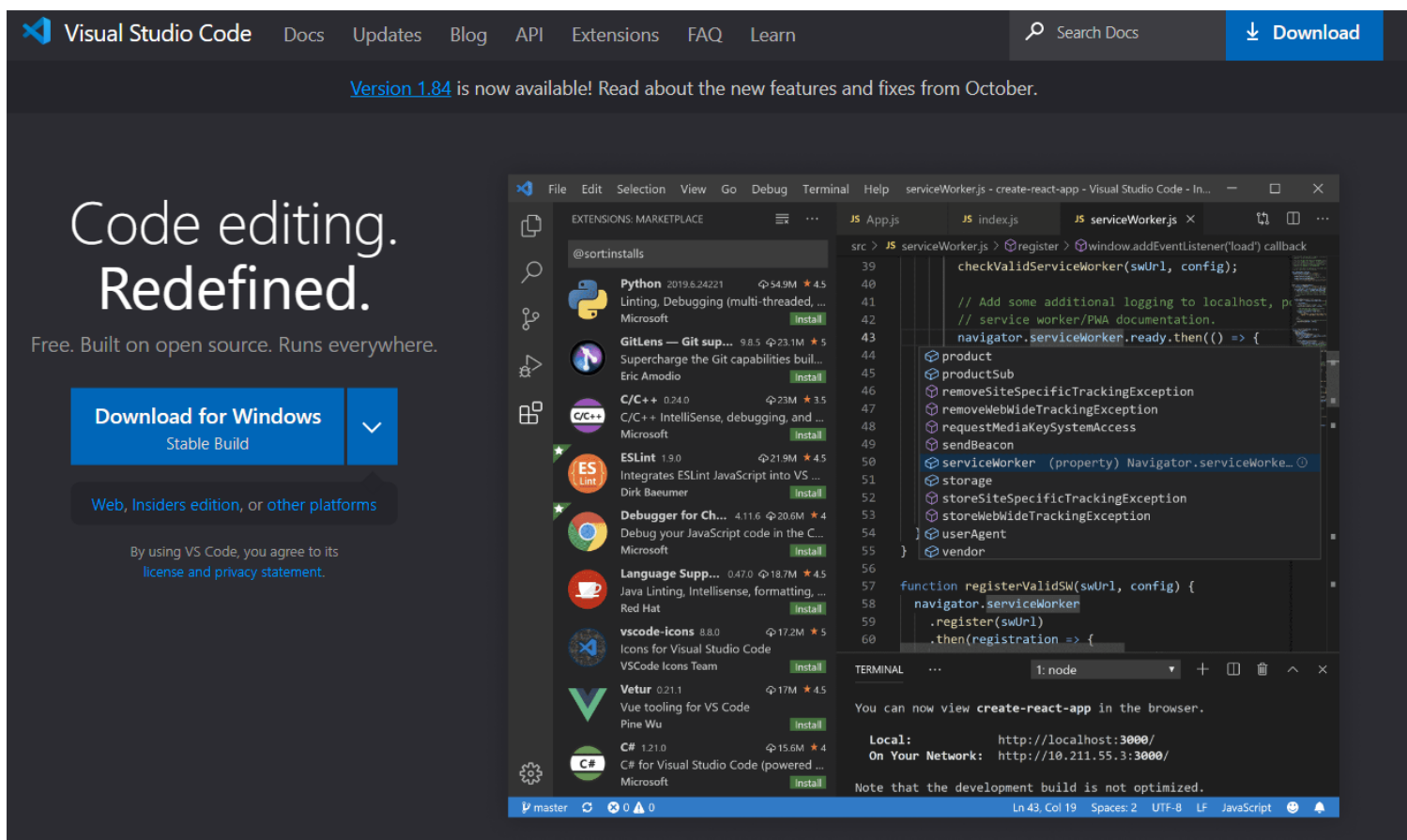
[Install .NET SDK for Linux.](#)



[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, 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 Verdanz.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"](#).

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

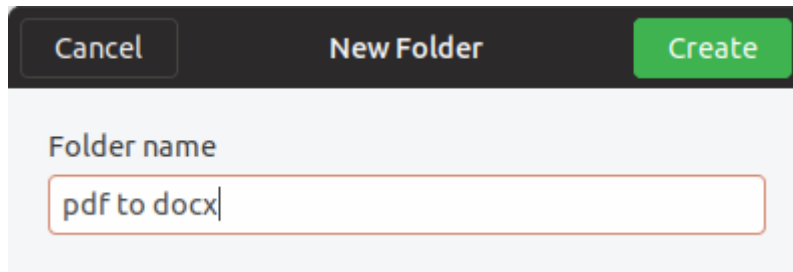
[Get Started with C# and Visual Studio Code](#)

Not only is possible to create .NET 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.

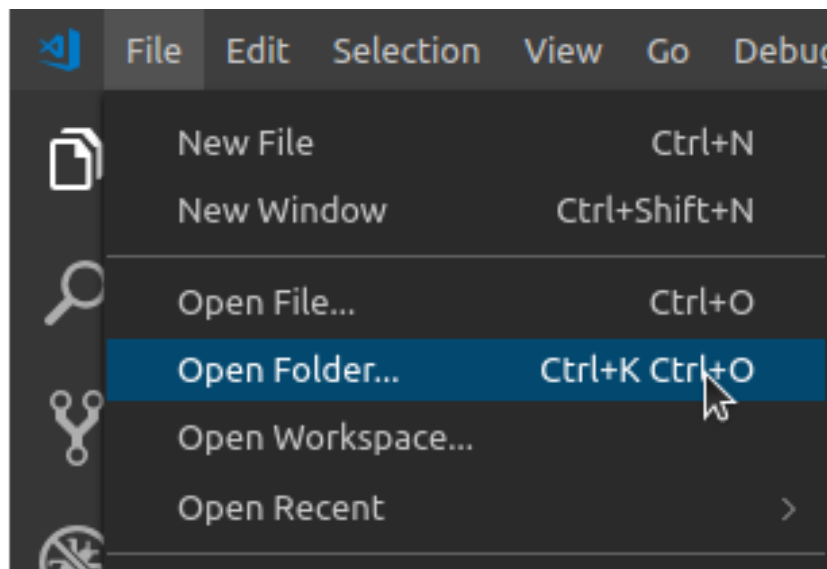
## 2. Creating “Convert PDF to DOCX” application

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

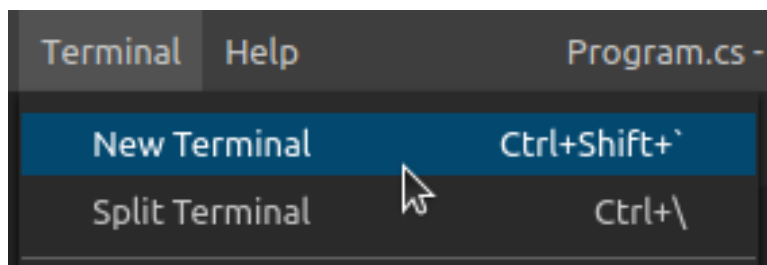
For example, let’s create the folder “**pdf to docx**” on Desktop (Right click-> New Folder):



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

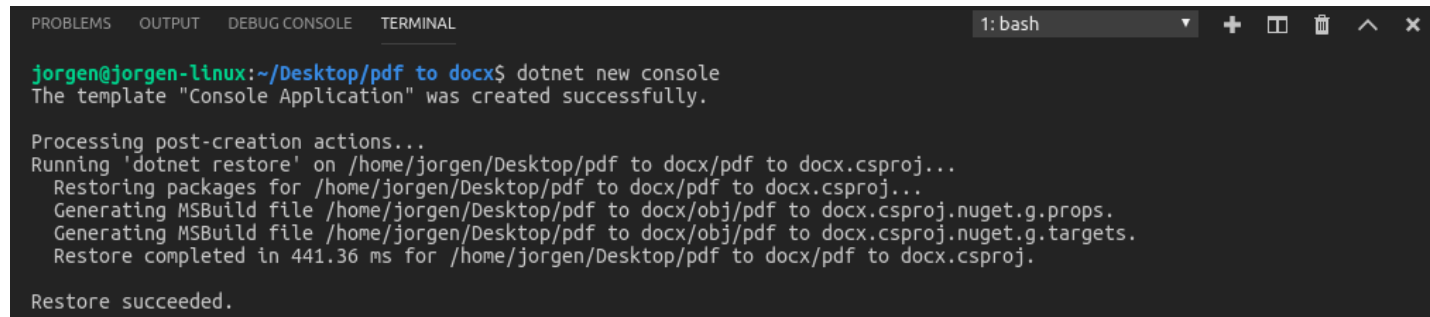


Now, open the integrated console – the Terminal: follow to the menu **Terminal -> New Terminal** (or press Ctrl+Shift+`):



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

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



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash
jorgen@jorgen-linux:~/Desktop/pdf to docx$ dotnet new console
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on /home/jorgen/Desktop/pdf to docx/pdf to docx.csproj...
  Restoring packages for /home/jorgen/Desktop/pdf to docx/pdf to docx.csproj...
  Generating MSBuild file /home/jorgen/Desktop/pdf to docx/obj/pdf to docx.csproj.nuget.g.props.
  Generating MSBuild file /home/jorgen/Desktop/pdf to docx/obj/pdf to docx.csproj.nuget.g.targets.
  Restore completed in 441.36 ms for /home/jorgen/Desktop/pdf to docx/pdf to docx.csproj.

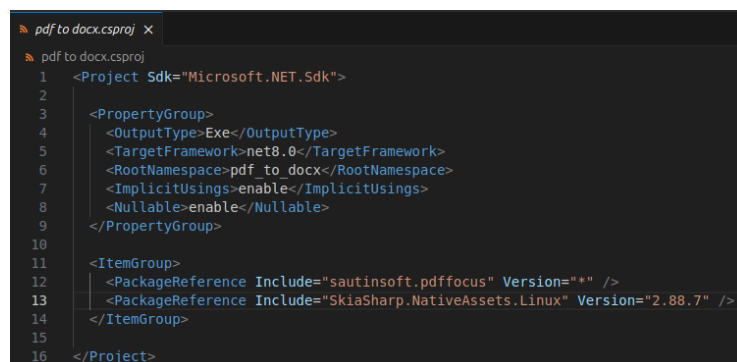
Restore succeeded.
```

Now we are going to modify this simple application 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.pdfdfocus** assembly using Nuget or the library SautinSoft.PdfFocus.dll with additional references.

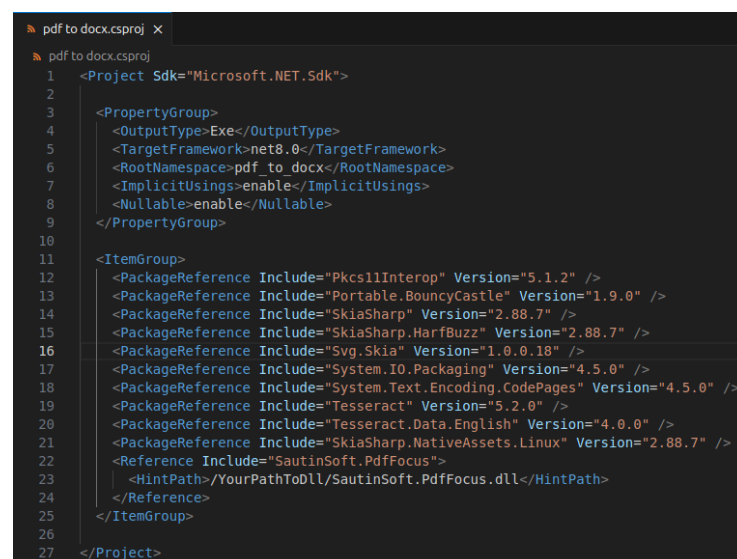
In order to do it, follow to the **Explorer** and open project file "**pdf to docx.csproj**":

In the first case (NuGet):



```
pdf to docx.csproj x
pdf to docx.csproj
1 <Project Sdk="Microsoft.NET.Sdk">
2
3   <PropertyGroup>
4     <OutputType>Exe</OutputType>
5     <TargetFramework>net8.0</TargetFramework>
6     <RootNamespace>pdf_to_docx</RootNamespace>
7     <ImplicitUsings>enable</ImplicitUsings>
8     <Nullable>enable</Nullable>
9   </PropertyGroup>
10
11   <ItemGroup>
12     <PackageReference Include="sautinsoft.pdfdfocus" Version="*" />
13     <PackageReference Include="SkiaSharp.NativeAssets.Linux" Version="2.88.7" />
14   </ItemGroup>
15
16 </Project>
```

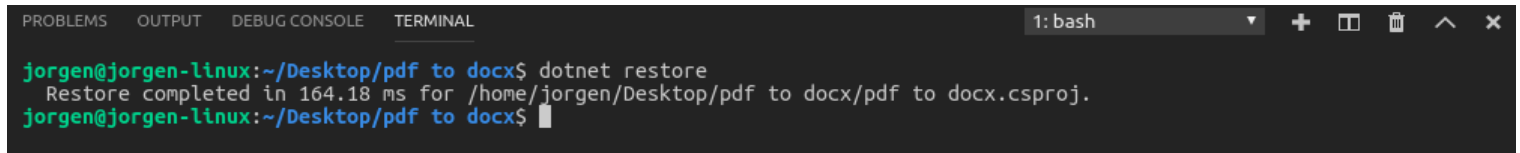
In the second case (SautinSoft.PdfFocus.dll):



```
pdf to docx.csproj x
pdf to docx.csproj
1 <Project Sdk="Microsoft.NET.Sdk">
2
3   <PropertyGroup>
4     <OutputType>Exe</OutputType>
5     <TargetFramework>net8.0</TargetFramework>
6     <RootNamespace>pdf_to_docx</RootNamespace>
7     <ImplicitUsings>enable</ImplicitUsings>
8     <Nullable>enable</Nullable>
9   </PropertyGroup>
10
11   <ItemGroup>
12     <PackageReference Include="Pkcs11Interop" Version="5.1.2" />
13     <PackageReference Include="Portable.BouncyCastle" Version="1.9.0" />
14     <PackageReference Include="SkiaSharp" Version="2.88.7" />
15     <PackageReference Include="SkiaSharp.HarfBuzz" Version="2.88.7" />
16     <PackageReference Include="Svg.Skia" Version="1.0.0.18" />
17     <PackageReference Include="System.IO.Packaging" Version="4.5.0" />
18     <PackageReference Include="System.Text.Encoding.CodePages" Version="4.5.0" />
19     <PackageReference Include="Tesseract" Version="5.2.0" />
20     <PackageReference Include="Tesseract.Data.English" Version="4.0.0" />
21     <PackageReference Include="SkiaSharp.NativeAssets.Linux" Version="2.88.7" />
22     <Reference Include="SautinSoft.PdfFocus">
23       <HintPath>YourPathToDll\SautinSoft.PdfFocus.dll</HintPath>
24     </Reference>
25   </ItemGroup>
26
27 </Project>
```

At once as we've added the package references, we have to save the **"pdf to docx.csproj"** and restore the added packages.

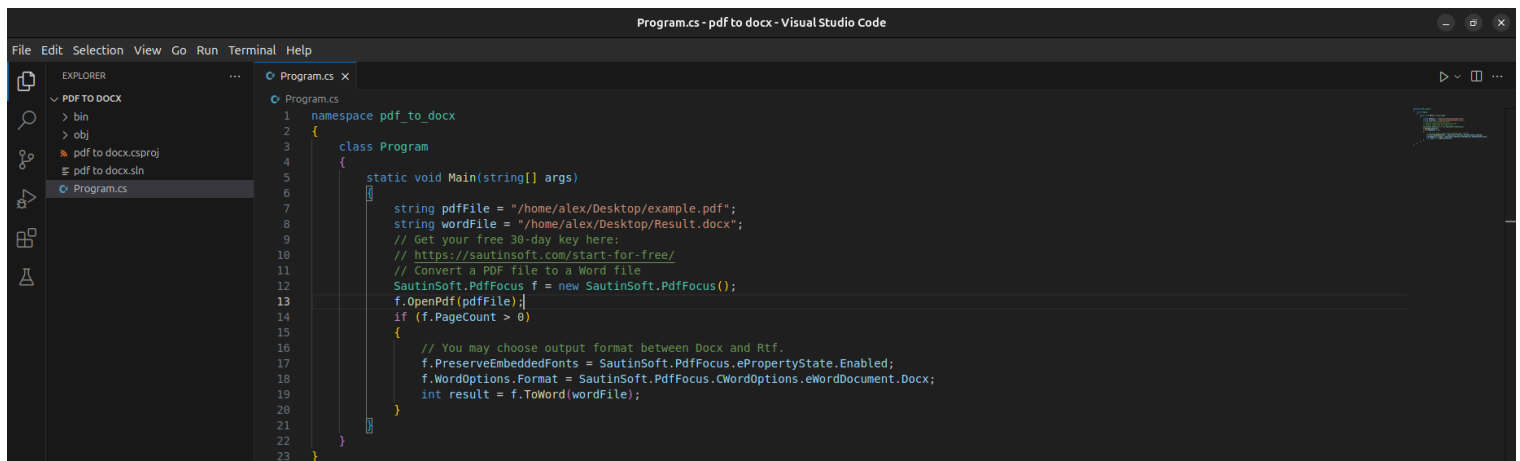
Follow to the **Terminal** and type the command: **dotnet restore**



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash
jorgen@jorgen-linux:~/Desktop/pdf to docx$ dotnet restore
Restore completed in 164.18 ms for /home/jorgen/Desktop/pdf to docx/pdf to docx.csproj.
jorgen@jorgen-linux:~/Desktop/pdf to docx$
```

Good, now our application has all the references 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:



```
Program.cs - pdf to docx - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
PDF TO DOCX
> bin
> obj
pdf to docx.csproj
pdf to docx.sln
Program.cs
Program.cs
1 namespace pdf_to_docx
2 {
3     class Program
4     {
5         static void Main(string[] args)
6         {
7             string pdfFile = "/home/alex/Desktop/example.pdf";
8             string wordFile = "/home/alex/Desktop/Result.docx";
9             // Get your free 30-day key here:
10            // https://sautinsoft.com/start-for-free/
11            // Convert a PDF file to a Word file
12            SautinSoft.PdfFocus f = new SautinSoft.PdfFocus();
13            f.OpenPdf(pdfFile);
14            if (f.PageCount > 0)
15            {
16                // You may choose output format between Docx and Rtf.
17                f.PreserveEmbeddedFonts = SautinSoft.PdfFocus.ePropertyState.Enabled;
18                f.WordOptions.Format = SautinSoft.PdfFocus.CWordOptions.eWordDocument.Docx;
19                int result = f.ToWord(wordFile);
20            }
21        }
22    }
23 }
```

## The code:

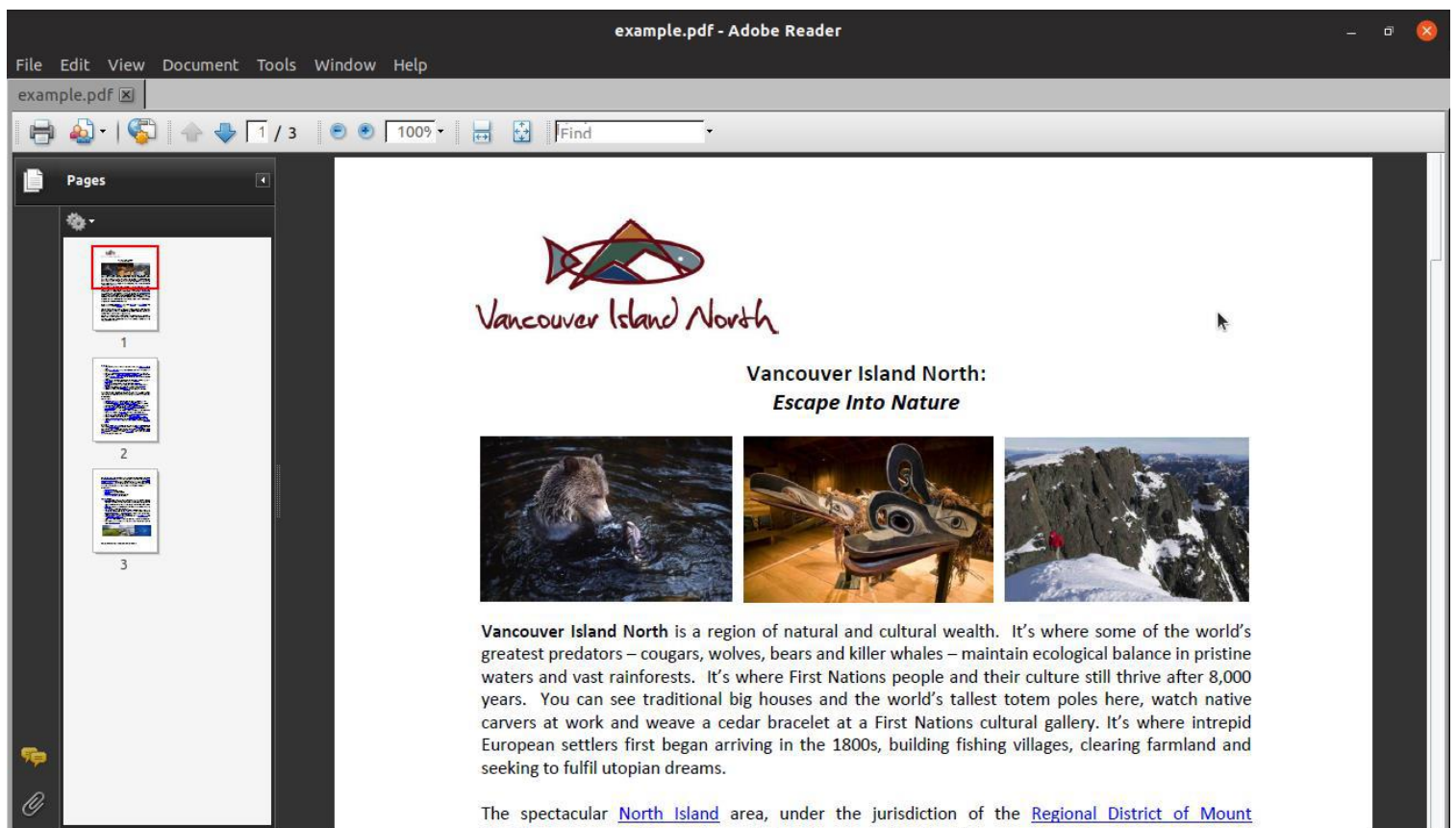
```
namespace pdf_to_docx
{
    class Program
    {
        static void Main(string[] args)
        {
            string pdfFile = "/home/alex/Desktop/example.pdf";
            string wordFile = "/home/alex/Desktop/Result.docx";
            // Get your free key here:
            // https://sautinsoft.com/start-for-free/
            // Convert a PDF file to a Word file
            SautinSoft.PdfFocus f = new SautinSoft.PdfFocus();
            f.OpenPdf(pdfFile);
            if (f.PageCount > 0)
            {
                // You may choose output format between Docx and Rtf.
                f.WordOptions.Format = SautinSoft.PdfFocus.CWordOptions.eWordDocument.Docx;
                int result = f.ToWord(wordFile);
            }
        }
    }
}
```



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 see its contents:



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

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
jorgen@jorgen-linux:~/Desktop/pdf to docx$ dotnet run
Converting successfully!
jorgen@jorgen-linux:~/Desktop/pdf to docx$
```

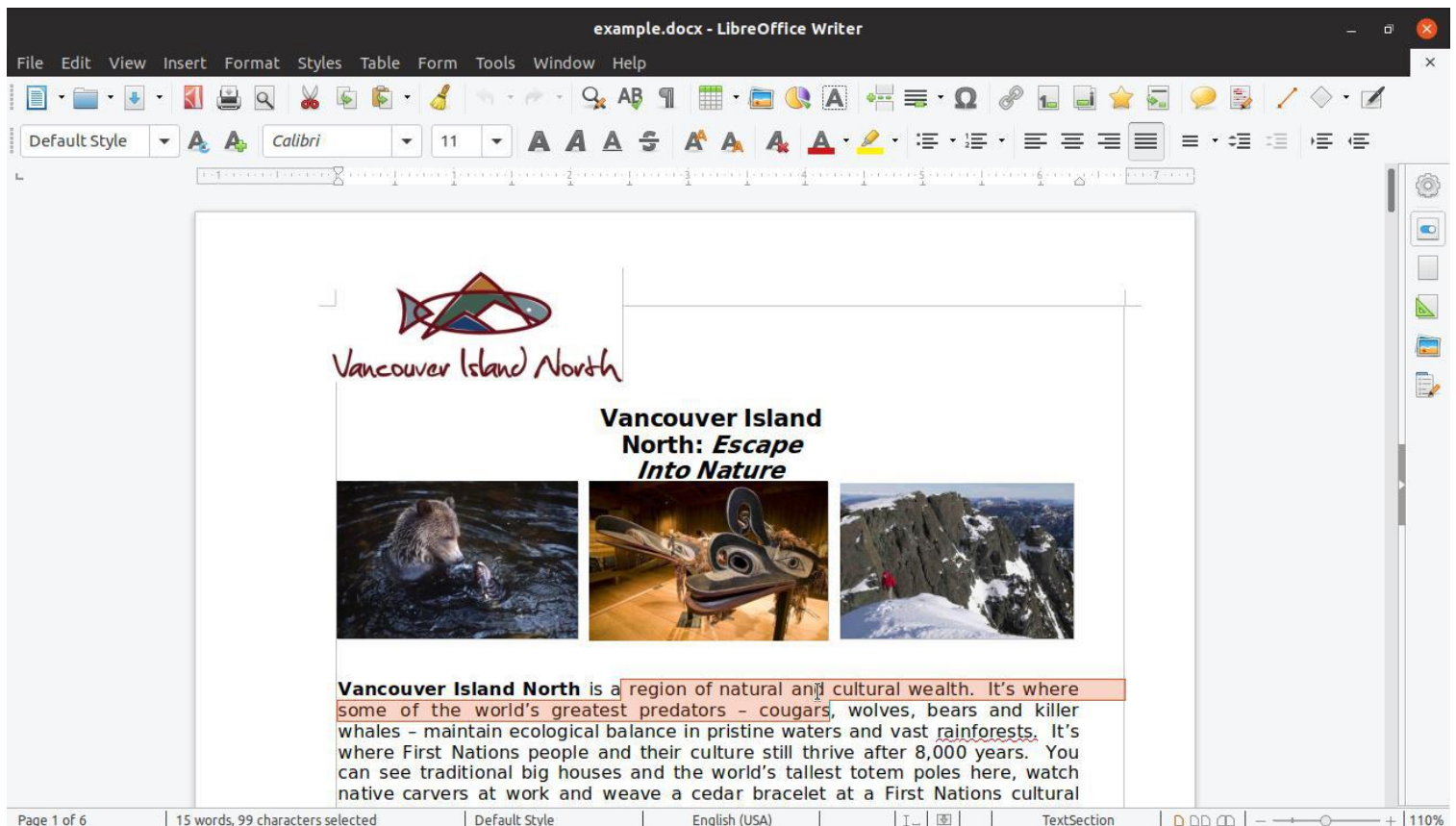


If you don't see any exceptions, everything is fine and we can check the result produced by The [PDF Focus .Net](#) library.

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!

If you have any troubles or need extra code, or help, don't hesitate to ask our SautinSoft Team at [support@sautinsoft.com](mailto:support@sautinsoft.com)!