

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 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.](#)

Please check the current version of .Net Core (2.X, 3.X)

Windows

Linux

macOS

.NET
Core

.NET Core 2.2

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

Build Apps ⓘ

Install .NET Core SDK

Run Apps ⓘ

Install .NET Core Runtime

[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

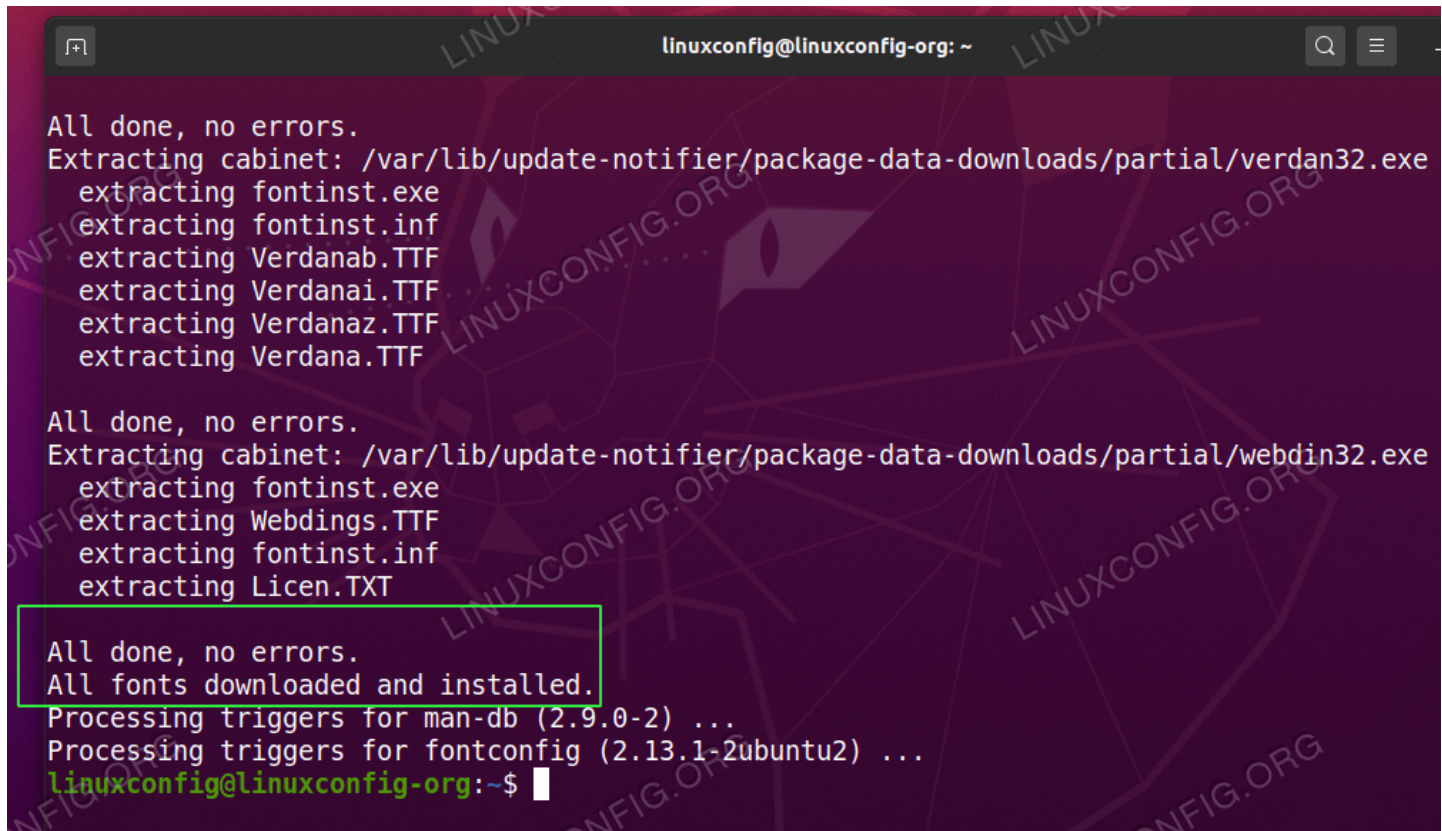
```
# /etc/rc.d/init.d/xfs restart
```

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
```



```
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 Verdana.TTF  
  extracting Verdana.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:

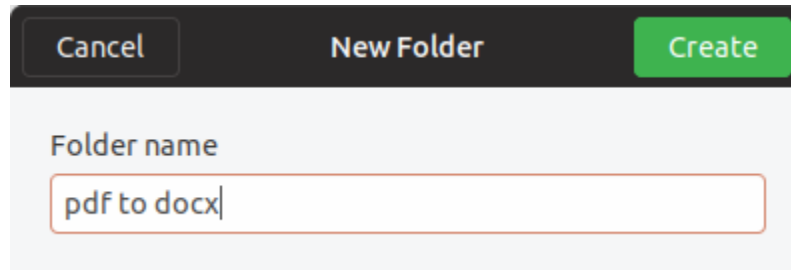
[Get Started with C# and Visual Studio Code](#)

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 2017.

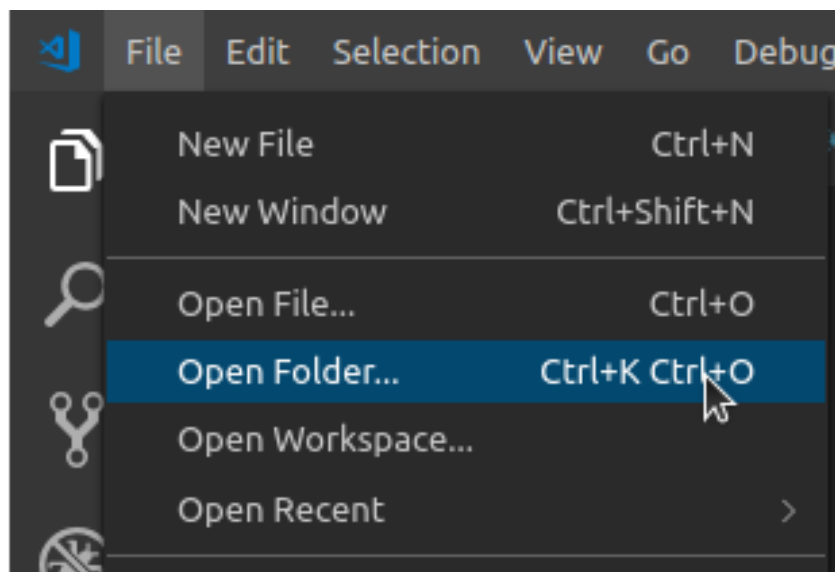
2. Creating “Convert PDF to DOCX” application

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

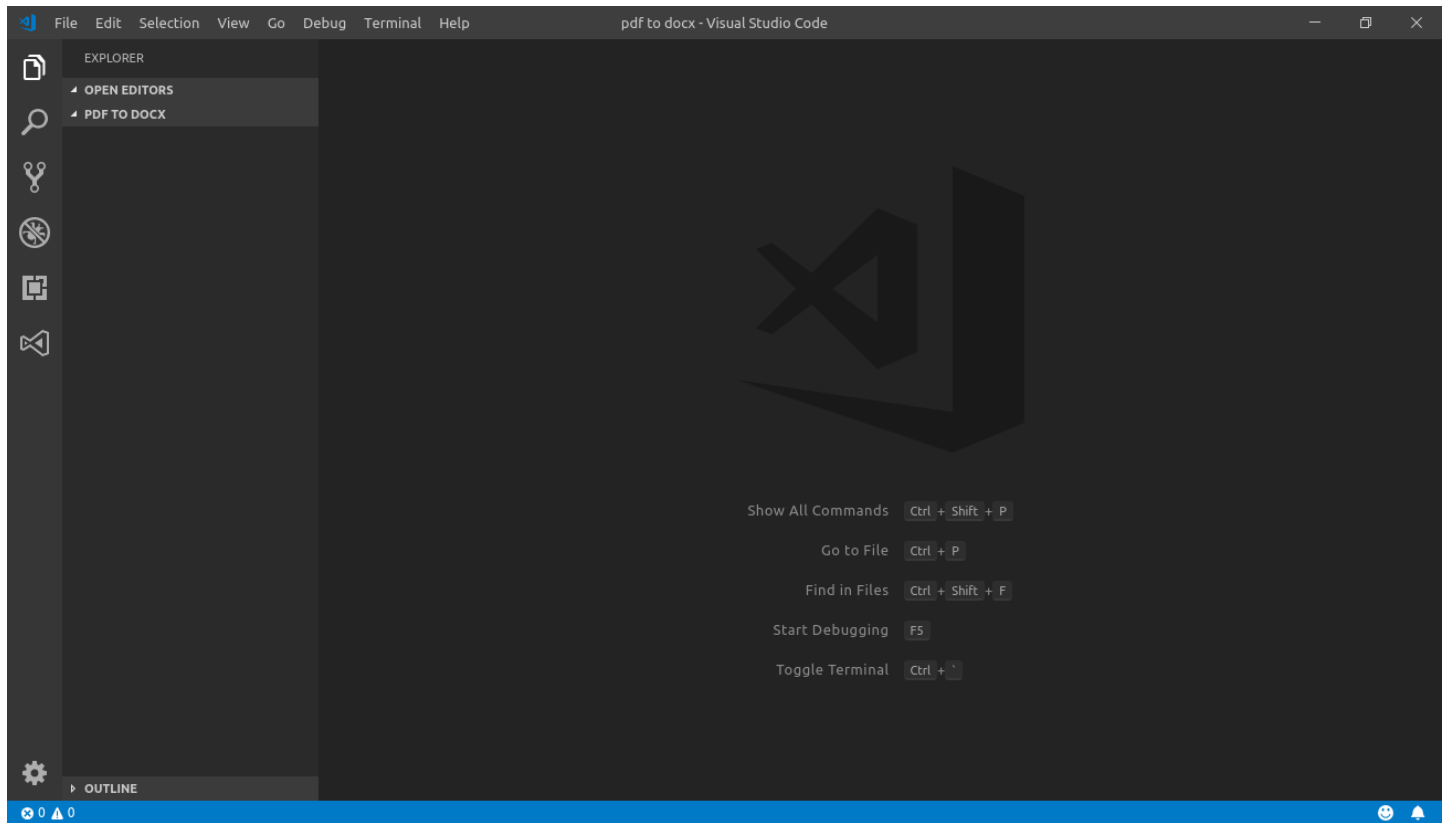
For example, let’s create the folder “**pdf to docx**” on Desktop (Right click-> New Folder):



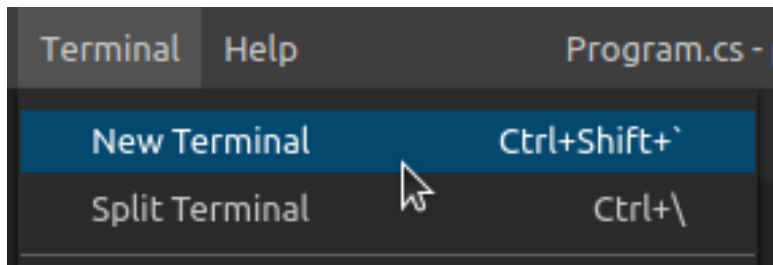
Open VS Code and click in the menu **File->Open Folder**. From the dialog, open the folder you’ve created previously:



Next you will see the similar screen:

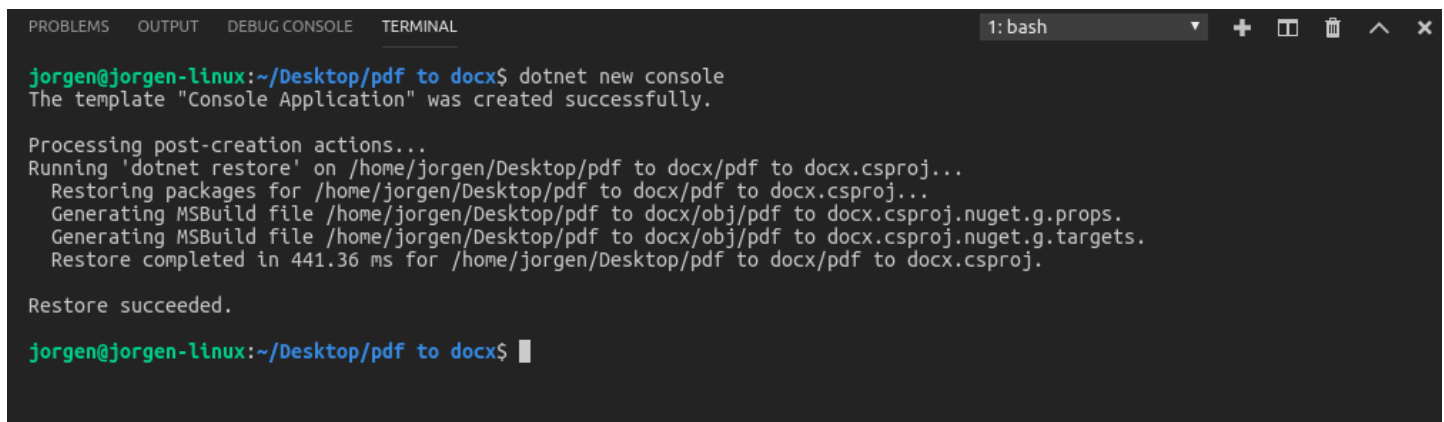


Now, open the integrated console – the Terminal: follow to the menu **Terminal** -> **New Terminal** (or press Ctrl+Shift+`):



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***

```
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.

jorgen@jorgen-linux:~/Desktop/pdf to docx$ dotnet run
Hello World!
jorgen@jorgen-linux:~/Desktop/pdf to docx$
```

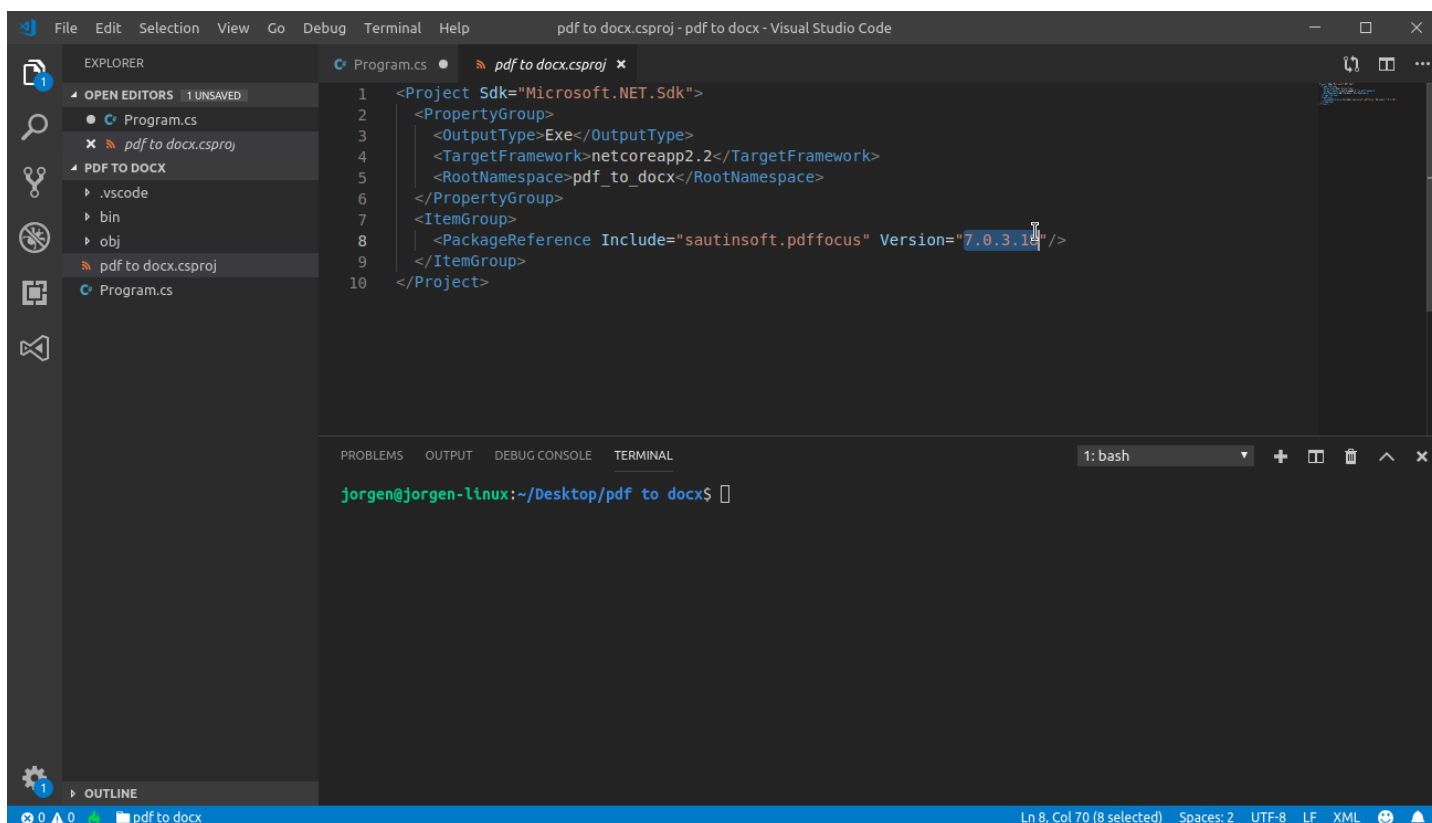
You can see the typical “Hello world!” message.

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.pdffocus*** 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:



```
File Edit Selection View Go Debug Terminal Help pdf to docx.csproj - pdf to docx - Visual Studio Code
EXPLORER
OPEN EDITORS 1 UNSAVED
  Program.cs
  pdf to docx.csproj
PDF TO DOCX
  .vscode
  bin
  obj
  pdf to docx.csproj
  Program.cs
1 <Project Sdk="Microsoft.NET.Sdk">
2   <PropertyGroup>
3     <OutputType>Exe</OutputType>
4     <TargetFramework>netcoreapp2.2</TargetFramework>
5     <RootNamespace>pdf_to_docx</RootNamespace>
6   </PropertyGroup>
7   <ItemGroup>
8     <PackageReference Include="sautinsoft.pdffocus" Version="7.0.3.14" />
9   </ItemGroup>
10  </Project>
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash
jorgen@jorgen-linux:~/Desktop/pdf to docx$
```

Add these lines into the file "**pdf to docx.csproj**":

```
<ItemGroup>  
    <PackageReference Include="sautinsoft.pdffocus" Version="7.0.3.18" />  
</ItemGroup>
```

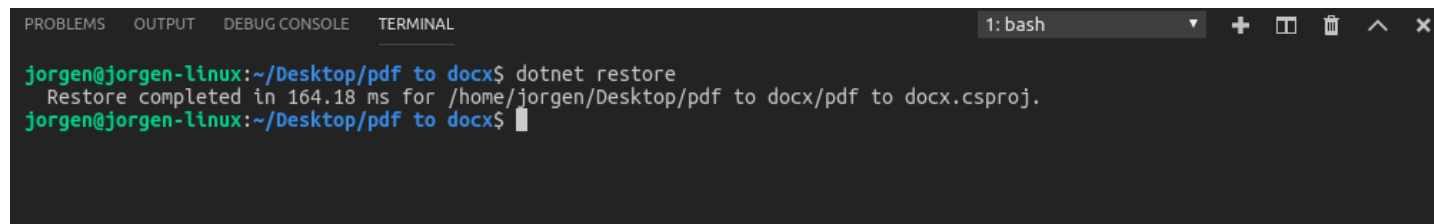
It's the reference to **sautinsoft.pdffocus** package from Nuget.

At the moment of writing this manual, the latest version of **sautinsoft.pdffocus** was 7.0.3.18. But you may specify the latest version, to know what is the latest, follow:

<https://www.nuget.org/packages/sautinsoft.pdffocus/>

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**



```
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$
```

Very important!

There are a lot of Linux varieties: Suse, Fedora, Debian, Ubuntu, etc.

In addition, there are cloud platforms: AWS Lambda, Google Cloud, Azure, Apex, etc.

Because our dll works with Graphics and using GDI+ API, you need to check, that your system has [System.Drawing.Common](#) is the graphics library which ships as part of .NET Core. On macOS and Linux, it uses [libgdiplus](#) as its back-end.

There are existing ways to install libgdiplus on macOS and Linux. On macOS, you can use the [mono-libgdiplus](#) Homebrew package; on Ubuntu Linux, you can install the [libgdiplus-dev](#) package.

Using libgdiplus on Ubuntu Linux

You can install libgdiplus on Ubuntu Linux using the Quamotion PPA. Follow these steps:

```
sudo add-apt-repository ppa:quamotion/ppa
sudo apt-get update
sudo apt-get install -y libgdiplus
```

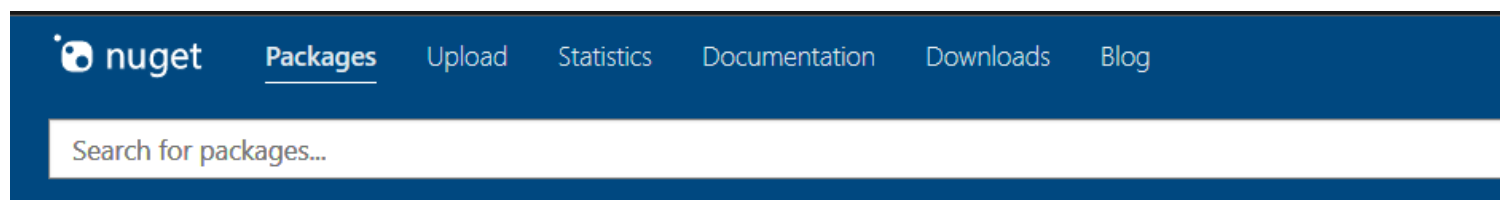
Using libgdiplus on macOS

On macOS, add a reference to the [runtime.osx.10.10-x64.CoreCompat.System.Drawing](#) package:

```
dotnet add package runtime.osx.10.10-x64.CoreCompat.System.Drawing
```

If you have installed LibGdiPlus' dll and it still doesn't work. Please add (update) this string in your project:

```
<PackageReference Include="runtime.osx.10.10-x64.CoreCompat.System.Drawing"
Version="5.23.62"/>
```



runtime.osx.10.10-x64.CoreCompat.System.
Drawing 5.8.64

Allows you to use System.Drawing on OS X

Package Manager

.NET CLI

PackageReference

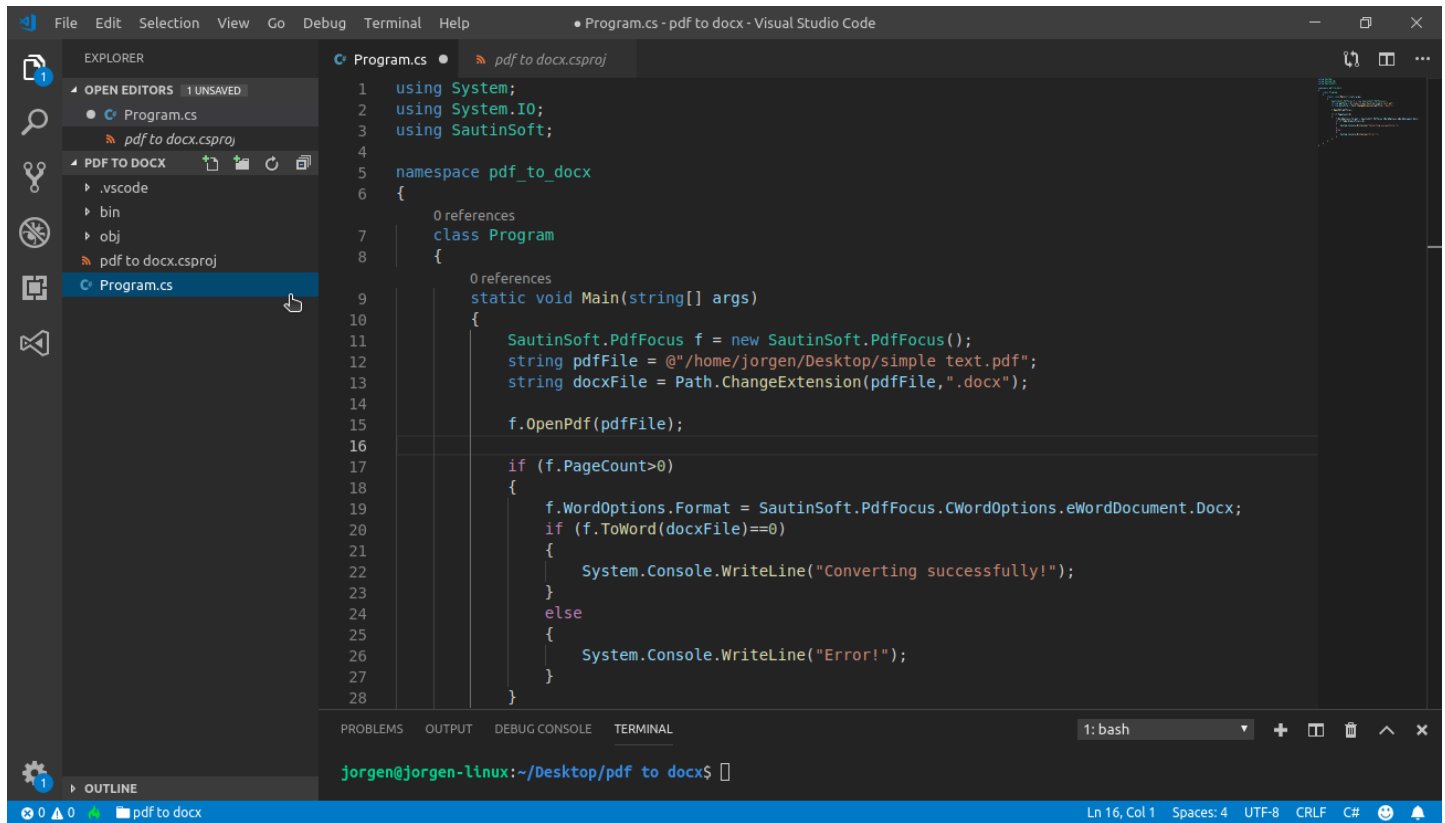
Paket CLI

```
PM> Install-Package runtime.osx.10.10-x64.CoreCompat.System.Drawing -Version 5.8.64
```



Good, now our application has the reference to **sautinsoft.pdffocus** 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:



```
1 using System;
2 using System.IO;
3 using SautinSoft;
4
5 namespace pdf_to_docx
6 {
7     0 references
8     class Program
9     {
10         0 references
11         static void Main(string[] args)
12         {
13             SautinSoft.PdfFocus f = new SautinSoft.PdfFocus();
14             string pdfFile = @"/home/jorgen/Desktop/simple text.pdf";
15             string docxFile = Path.ChangeExtension(pdfFile, ".docx");
16
17             f.OpenPdf(pdfFile);
18
19             if (f.PageCount>0)
20             {
21                 f.WordOptions.Format = SautinSoft.PdfFocus.CWordOptions.eWordDocument.Docx;
22                 if (f.ToWord(docxFile)==0)
23                 {
24                     System.Console.WriteLine("Converting successfully!");
25                 }
26                 else
27                 {
28                     System.Console.WriteLine("Error!");
29                 }
30             }
31         }
32     }
33 }
```

The new code:

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

namespace pdf_to_docx
{
    class Program
    {
        static void Main(string[] args)
        {
            SautinSoft.PdfFocus f = new SautinSoft.PdfFocus();
            string pdfFile = @"/home/jorgen/Desktop/simple text.pdf";
            string docxFile = Path.ChangeExtension(pdfFile, ".docx");

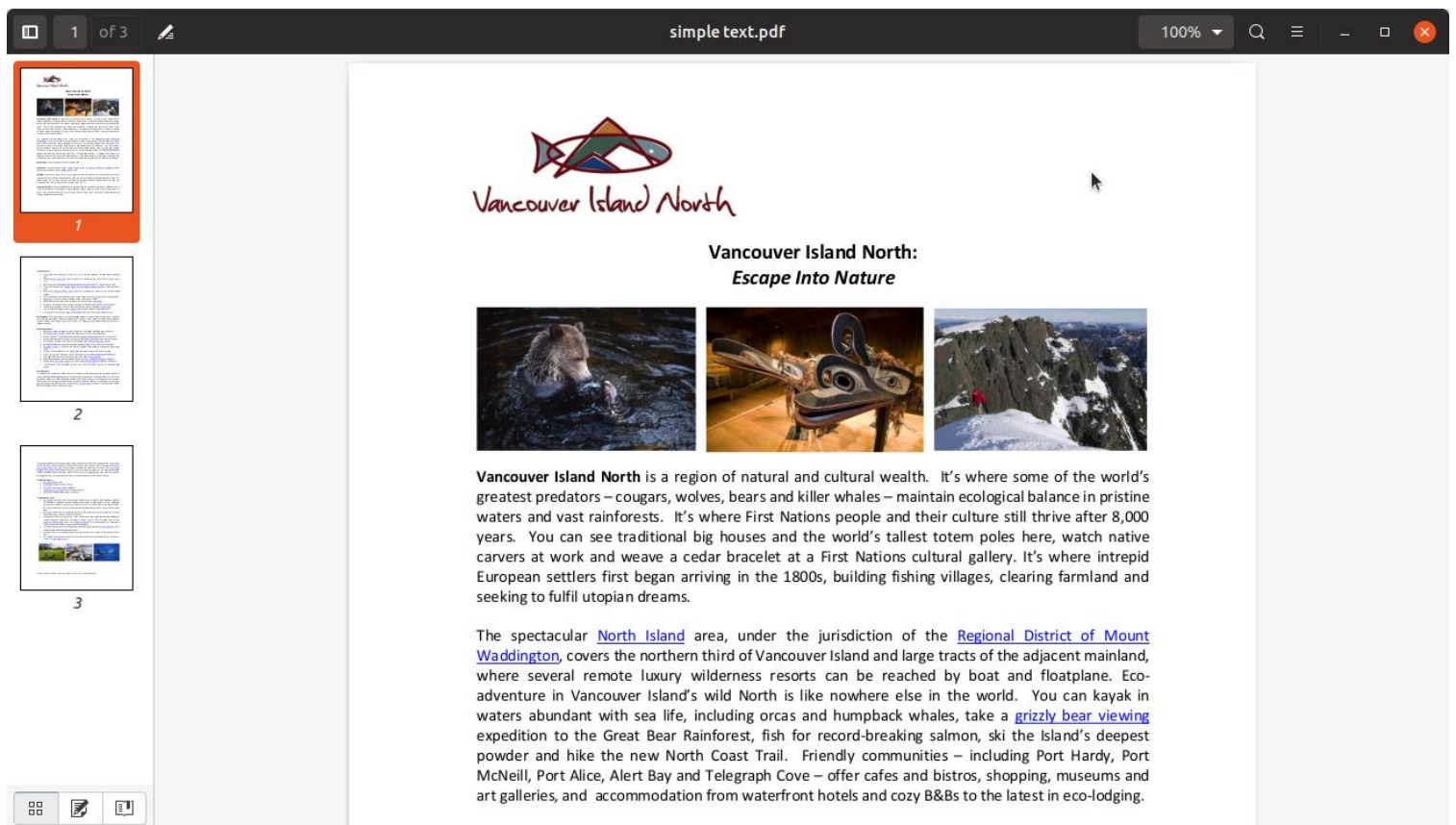
            f.OpenPdf(pdfFile);

            if (f.PageCount>0)
            {
                f.WordOptions.Format =
                SautinSoft.PdfFocus.CWordOptions.eWordDocument.Docx;
                if (f.ToWord(docxFile)==0)
                {
                    System.Console.WriteLine("Converting successfully!");
                }
                else
                {
                    System.Console.WriteLine("Error!");
                }
            }
        }
    }
}
```

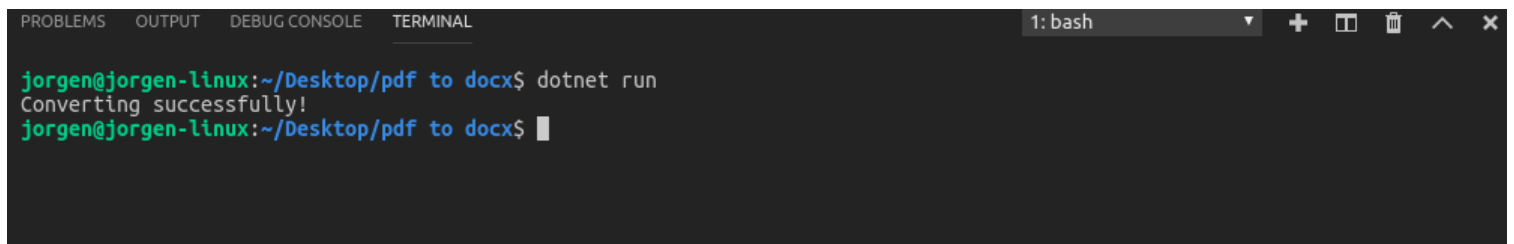
To make tests, we need an input PDF document. For our tests, let's place a PDF file with the name "simple text.pdf" at the Desktop.



If we open this file in the default PDF Viewer, we'll its contents:



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

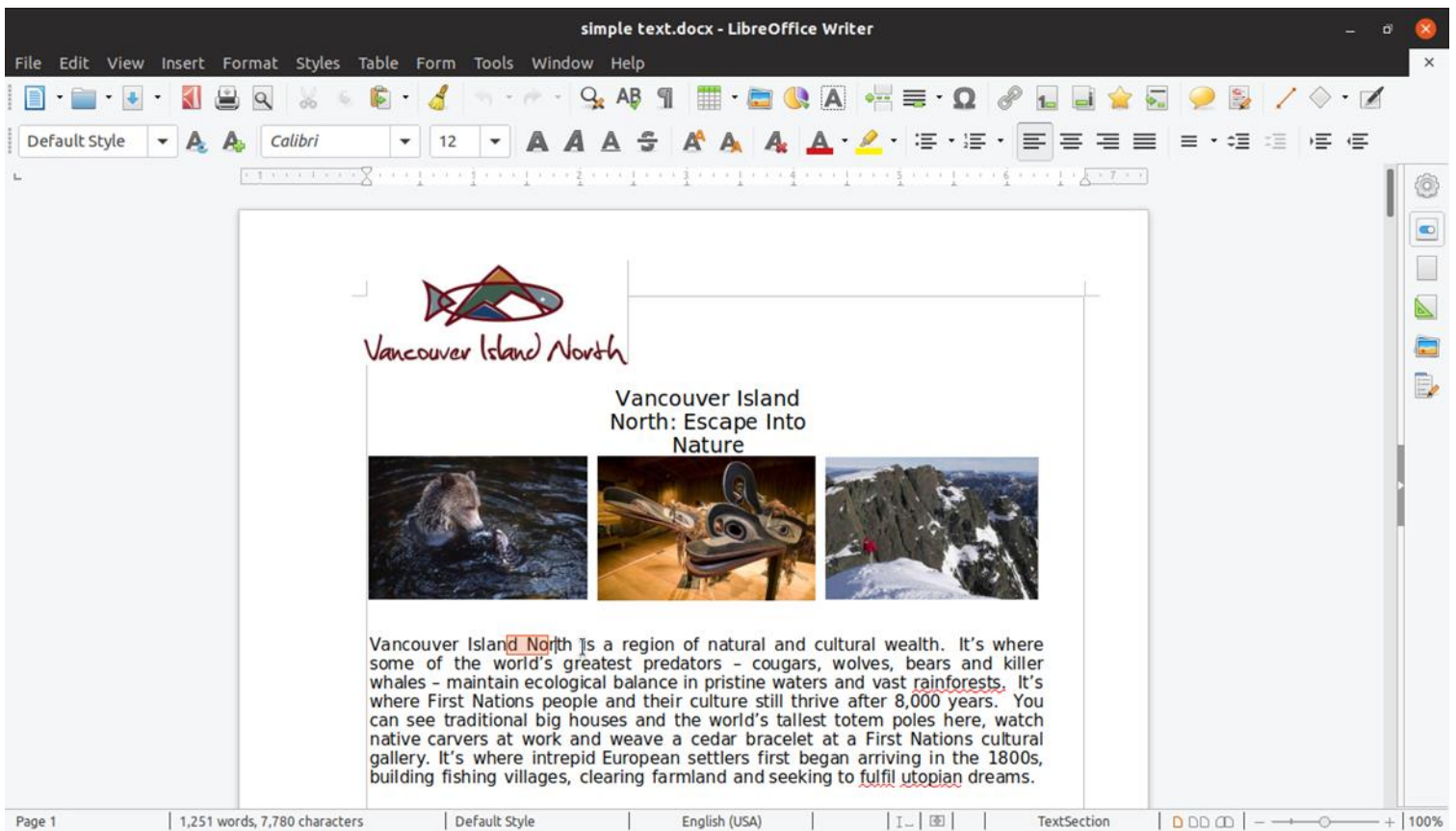


If you see the message “Converting successfully!”, everything is fine and we can check the result produced by the PDF Focus .Net library.

The new file “simple text.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.