



## PDFPrinting.NET Usage Guide

## Contents

What is PDFPrinting.NET? .....	4
Software requirements .....	4
Installation.....	5
Samples .....	6
<i>PdfrintLibraryTest32/64 bit</i> .....	6
<i>SimplePdfPrintLibraryTest32/64bit</i> .....	10
<i>PdfPrintLibraryMultiDocumentPrintTest32/64bit</i> .....	11
<i>PdfViewerNetDemo32/64 bit</i> .....	12
<i>Installation - Steps</i> .....	13
<i>Installation for print or convert to images functionality</i> .....	13
<i>Installation for PdfViewer functionality</i> .....	16
Available Shortcuts in PdfViewer Component .....	20
PdfPrint Examples.....	21
<i>Simple silent printing of PDF files</i> .....	21
<i>Silent printing with already installed Adobe Reader</i> .....	21
<i>Using PrinterSettings in Printing</i> .....	21
<i>Convert PDF documents into images</i> .....	22
<i>Working with password protected PDF documents</i> .....	23
<i>Custom print properties</i> .....	23
PdfViewer Examples .....	24
<i>Open Document with open document status</i> .....	24
<i>Toolbar options – change different Toolbar properties</i> .....	24
<i>Print and its events</i> .....	25
<i>Zoom functionality</i> .....	26
<i>PDF page navigation</i> .....	26
<i>Selection</i> .....	27
<i>Miscellaneous properties and functions</i> .....	27
<i>Search</i> .....	27
PdfPrint FAQ.....	28
<i>Where can I see a code sample for usage of the PDFPrinting.NET?</i> .....	28
<i>Can I use PDFPrinting.NET in windows service?</i> .....	28

<i>Can I use PDFPrinting.NET to print 2 pages to one page/ merge 2 pdf documents? .....</i>	<i>29</i>
<i>Can I use PDFPrinting.NET to print to file? .....</i>	<i>29</i>
<i>Are paper sources (input trays) and output trays supported? .....</i>	<i>30</i>
<i>Can I include PdfPrintingNet.dll in my setup file? (ClickOnce, MSI, ...).....</i>	<i>30</i>
<i>How page scaling in printing works?.....</i>	<i>31</i>
<i>Is PdfPrintingNet.dll signed with public key? .....</i>	<i>31</i>
<i>Can I merge pdfprint dll with ilmerge tool?.....</i>	<i>31</i>
<i>Can I convert images to pdf?.....</i>	<i>31</i>
<i>How can I license the software? What is needed in my redistributable application?.....</i>	<i>32</i>
<i>What are limitations of the demo version of pdfprint library? .....</i>	<i>32</i>
<i>What is the difference between Viewer, Print, and Full license? .....</i>	<i>32</i>
<b>PdfViewer component FAQ.....</b>	<b>33</b>
<i>How to license PdfViewer object? .....</i>	<i>33</i>
<i>When calling the Print() method on the Viewer component I get the print dialog. Can I silently print a document? .....</i>	<i>33</i>
<i>Can I remove some of the options on the toolbar of Viewer component?.....</i>	<i>33</i>
<i>Why is print button on the Viewer component grayed out? .....</i>	<i>33</i>
<i>How to add PdfViewer component on my Windows Form in design time from Visual Studio? .....</i>	<i>33</i>
<i>Does Viewer component respects PDF permissions? .....</i>	<i>34</i>
<i>What are limitations of the demo version of PdfViewer? .....</i>	<i>34</i>

## What is PDFPrinting.NET?

PDFPrinting.NET is a library which enables you to:

1. Print your PDF documents
2. Convert them into images with ease
3. Use our PdfViewer component for displaying PDF documents in yours .NET applications

As a standalone .NET assembly, just reference it in your product and start coding right away. The library does not depend on third party libraries, so you won't have to worry about any further licensing issues.

Also, you don't need to use Adobe Reader with our product, because our PDF printing function works perfectly on its own.

Reproduction of a PDF document to a printer ensures that any printed document will look like the digital one. Furthermore, PDF printing in C#/VB.Net is simple with our library, and it is compatible with the .NET framework and all CLR languages.

## Software requirements

Please take a look at the list below for the information on supported operating systems.

### Operating Systems

- Windows 2016 Server
- Windows 2012 R2 Server
- Windows 2012 Server
- Windows 2008 R2 Server
- Windows 2008 Server
- Windows 2003 Server
- Windows 10
- Windows 8, 8.1
- Windows 7
- Windows Vista
- Windows XP

**Note:** 32bit and 64bit operating systems are supported. PdfPrint requires .NET 2.0/3.5 or 4.0/4.5 framework to function properly. In case you experience any issues, please make sure that .NET is installed.

## Installation

Please run the following download link:

[Download link](#)

After extracting files, you will notice the following:

- [PdfPrintingNet.dll](#) - pdfprinting.net library which you will use in your code. That is AnyCPU version.
- [PdfPrintingNet.chm](#) - help file with an explanation of all available PdfPrintingNet.dll properties and methods.
- [Samples folder](#) – demo applications which allow you to test our library easily. Every demo application has been compiled as 32 and 64-bit version.
- [Samples source folder](#) – source code for every demo application in Sample folder. We strongly suggest you review that code to see how to use PdfPrintingNet.dll properly.
- [Bin folder](#) – our library compiled as 32-bit, 64-bit, and AnyCPU version.

To use our library in your project, add a reference to our dll and:

1. For print/convert functionality, add to your code:

```
var pdfPrint = new PdfPrint("demoCompany", "demoKey");
```

2. As PdfViewer component adds to your code:

```
var pdfViewer = new PdfViewer();  
pdfViewer.SetLicenseInfo("your company name", "your license key");
```

Special notice: When you purchase the product you will get the licensed serial key.

## Samples

### *PdfrintLibraryTest32/64 bit*

PdfPrintLibraryTest32.exe / PdfPrintLibraryTest64.exe is a demo used for test our library.  
There are three tabs:

- Print pdf
- Print pdf with Adobe
- Save pdf as images

In the first tab you will notice all standard printing setup options to try out with your pdf file. This tab uses our standard printing engine.

The screenshot shows the 'TerminalWorks - PdfPrinting.Net Test - 32 bit ver. 3.0.5.6' window. The 'Print PDF' tab is selected. The interface includes a 'Select a PDF file:' section with a text box containing 'C:\pdf\pdf\_reference\_1-7.pdf' and a 'Password:' field. Below this are three tabs: 'Print PDF', 'Print PDF With Adobe', and 'Save PDF pages as images'. The 'Print PDF' tab contains several sections: 'Printer Settings' with options for printer name, number of copies, duplex type, color, collate, paper source, resolution, paper size, and orientation; 'Print range' with a text box for page ranges and radio buttons for even/odd page selection; 'Scale' with radio buttons for actual size, fit to margins, and shrink to margins; and 'Content Orientation' with checkboxes for auto rotate and centered. At the bottom, there are checkboxes for 'Send file as byte[]', 'Use Watermark', and a 'Print' button.

It is possible to set watermark.

**Set Watermark** [X]

Image Path:  
c:\test\watermark\_demo.jpg [...]

Watermark Text  
Text:  
Watermark test

Selected Font: Microsoft Sans Serif, 12 pt  
[Set Font]

Selected Color: Black  
[Set Color]

Opacity %: 100

Position  
Rotation Angle: 45 ☒ Use Relative % Position X: 45 Y: 45

Preview  
Paper Width: 8.27 Paper height: 11.69 [Preview]

Paper size is in inches. For example A4 is 8.27" X 11.69", Letter is 8.5" x 11", ...

[OK] [Cancel]

In the second tab you can test printing using Adobe engine. Adobe engine is a wrapper which calls already installed Adobe Reader for printing. Adding watermark isn't supported.

TerminalWorks - PdfPrinting.Net Test - 32 bit ver. 3.0.5.6

Select a PDF file:  
C:\pdf\pdf\_reference\_1-7.pdf

Password:

Print PDF   Print PDF With Adobe   Save PDF pages as images

**Printer Settings**

Printer name: TSPrint Printer

Number of copies  
☒ Use printer default   or set number of copies: 1

Duplex type  
☒ Use printer default   ☐ Horizontal   ☒ Simplex   ☐ Vertical

Print in color or no?  
☒ Use printer default   ☒ Print in color

Collate  
☒ Use printer default   ☒ Collate

Paper source  
☒ Use printer default   Paper source: [PaperSource Automatically Select Kind=FormSource]

Printer resolution  
☒ Use printer default   Resolution: [PrinterResolution X=300 Y=300]

Paper size  
☒ Use printer default   Paper size: [PaperSize A4 Kind=A4 Height=1169 Width=827]

Paper orientation:  
☒ Use printer default   ☐ Landscape   ☒ Portrait

☐ Use defined Printer Settings   Set Printer Settings   ☐ Use Printer Preferences Dialog

**Print range**  
Print pages: (e.g. 1-3 or exact page number). Multiple ranges separated by "," aren't supported (e.g. 1-3.5). If not set, all pages will be printed.

**Scale**  
☐ Use actual size   ☒ Shrink to margins

**Content Orientation**  
☐ Auto Rotate (if set, this will override Paper orientation)

Print with Adobe



In the third tab you can test variety of options which are used in saving pdf as images.

TerminalWorks - PdfPrinting.Net Test - 32 bit ver. 3.0.5.6

Select a PDF file:  
C:\pdf\pdf\_reference\_1-7.pdf

Password:

Print PDF   Print PDF With Adobe   **Save PDF pages as images**

**Image settings**

Color Type  
☐ Black and white   ☐ Gray   ☒ As it is in PDF

Image quality: 100   *It has effect just in some image formats. (For example jpg)*

Zoom factor: 5

From page: 1   To page: 1

Type of image file is determined by filename extension. If extension is omitted than it will be saved as bmp.  
If creating images from more than one page, name of output file will be output image file name + page number.  
(Example: From page = 2, To Page = 4, outputfilename = test.jpg it will create files test2.jpg, test3.jpg, test4.jpg)

Output image file name:  
c:\test\test.jpg

X resolution: 200   Y resolution: 200

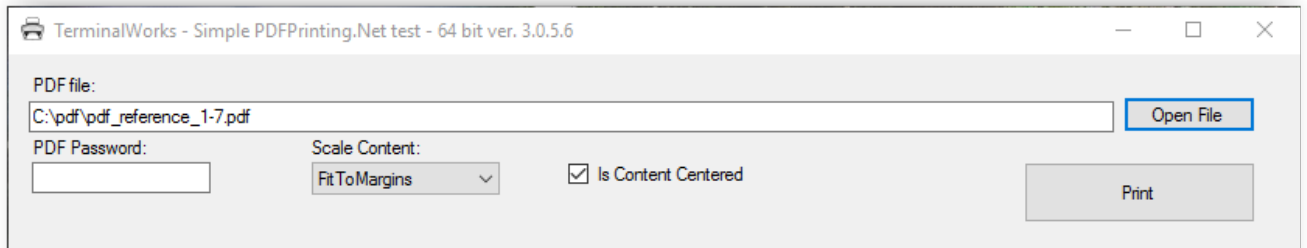
Multi page tiff file  
☐ Create multi page tiff file   Tiff Compression: LZW

☐ Send file as byte[]

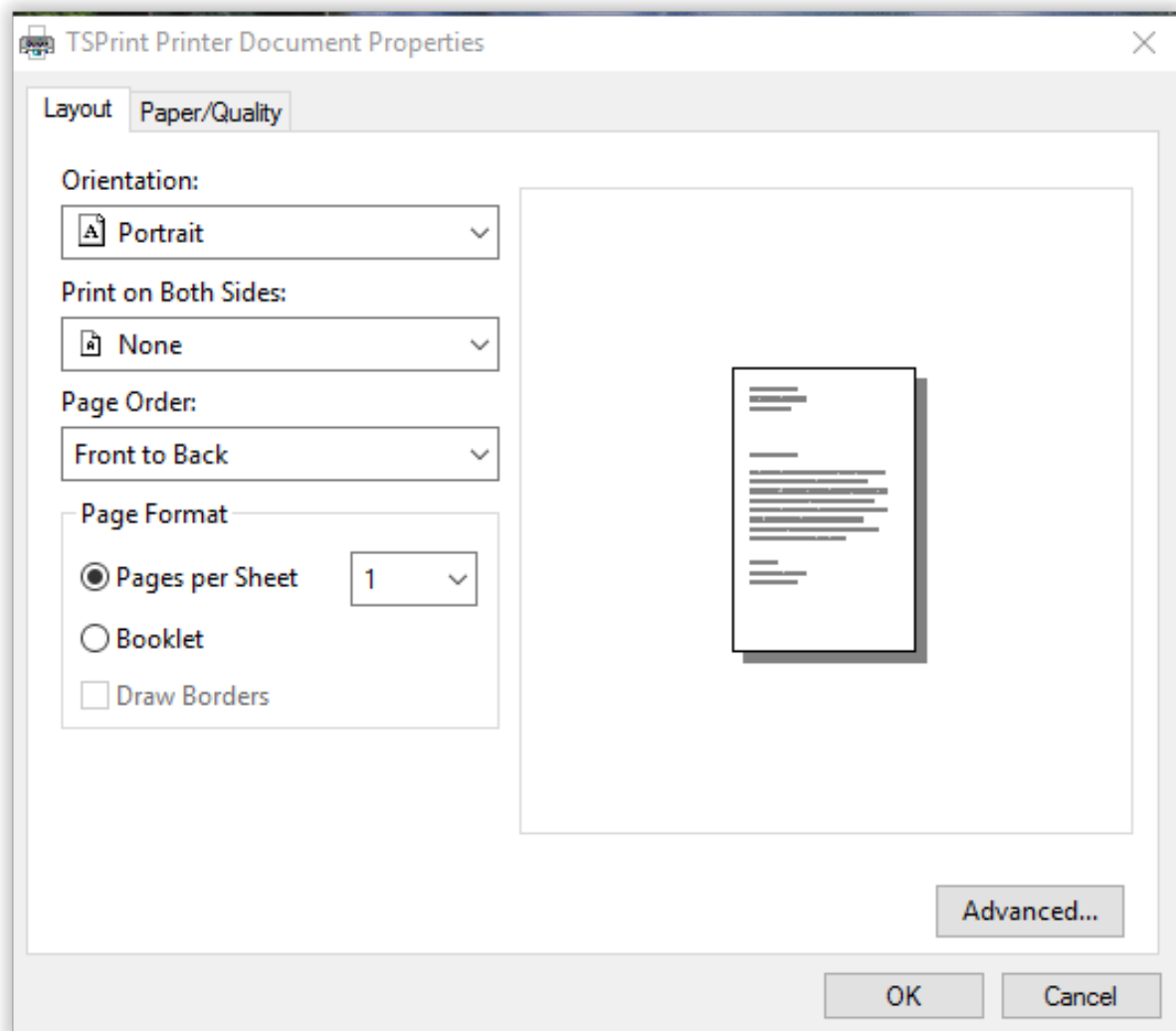
Save PDF page(s) as image(s)

## SimplePdfPrintLibraryTest32/64bit

SimplePdfPrintLibraryTest32.exe / SimplePdfPrintLibraryTest64.exe is a demo used for test our library. It is very simple and with just basic options.



Pressing Print button opens a Print dialog where you can set additional printer options.



## PdfPrintLibraryMultiDocumentPrintTest32/64bit

Multi document print as single printer job.

TerminalWorks PdfPrint MultiDocument Print Test ver. 32 bit ver. 3.0.5.6

Select up to 5 PDF documents for multi print as single printer job.

Printer name:

Job Name:

PDF document 1:  ... Properties

PDF document 2:  ... Properties

PDF document 3:  ... Properties

PDF document 4:  ... Properties

PDF document 5:  ... Properties

For every document it is possible to set its printer options.

Properties for PDF Document 1

**Printer Settings**

Print in color or no?  
☒ Use printer default ☒ Print in color

Paper orientation:  
☒ Use printer default ☐ Landscape ☒ Portrait

Paper source  
☒ Use printer default Paper source:

Printer resolution  
☒ Use printer default Resolution:

Paper size  
☒ Use printer default Paper size:

☐ Use defined Printer Settings

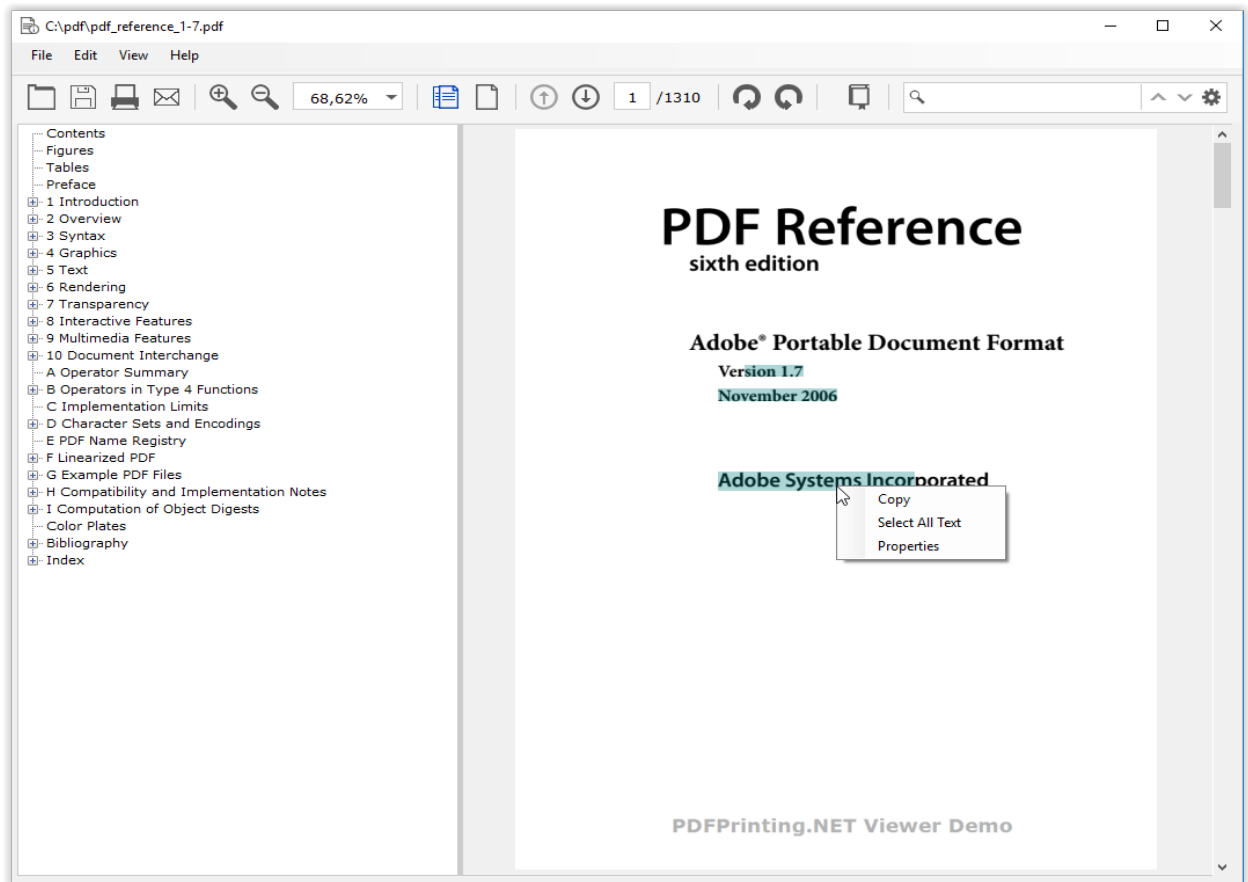
**Print range**  
Print pages: (e.g. 1-3;4,7 - if empty it will print all pages). Reverse range is also supported (e.g. 5-2)  
  
☒ Print even and odd pages ☐ Print only even pages ☐ Print only odd pages

**Scale**  
☐ Use actual size ☐ Fit to margins ☒ Shrink to margins

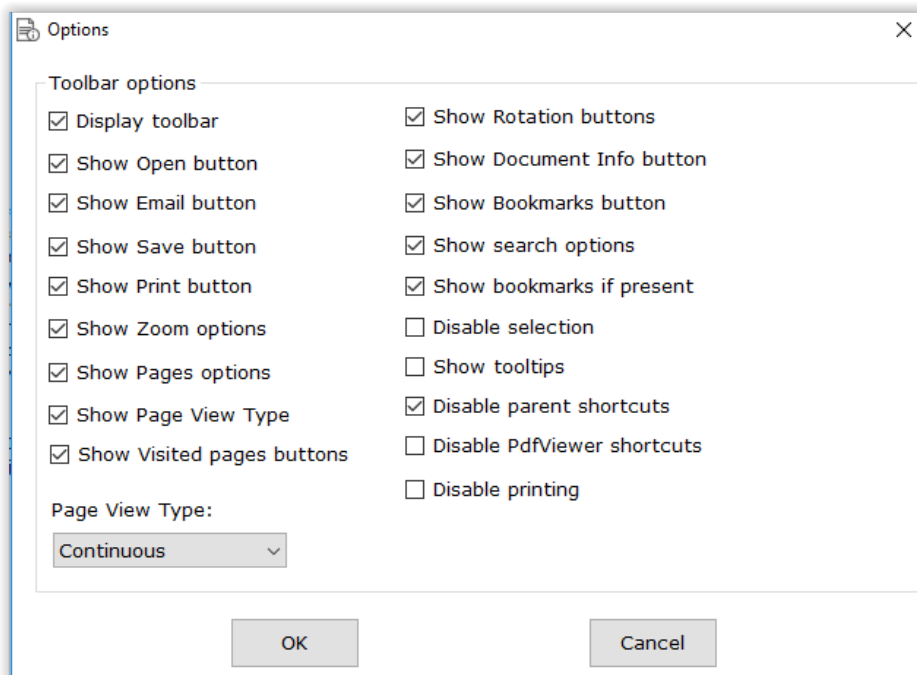
**Content Orientation**  
☐ Auto Rotate (if set, this will override Paper orientation) ☒ Centered

☐ Send file as byte[] ☐ Use Watermark  Password:

## PdfViewerNetDemo32/64 bit



It is possible to hide toolbar or just some of its options.

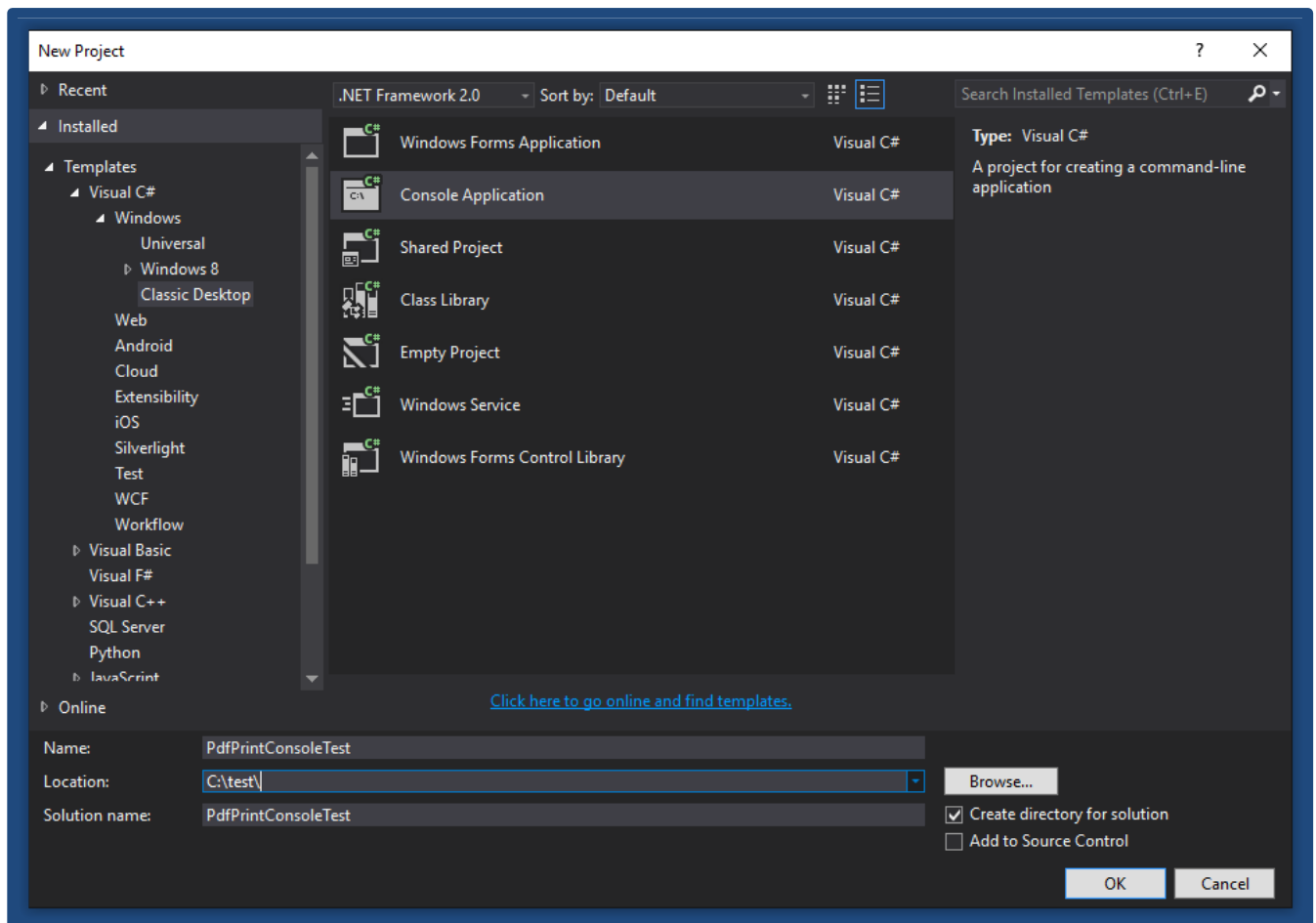


## Installation - Steps

In this tutorial, we are using C#, but it will also work for other supported .NET languages.

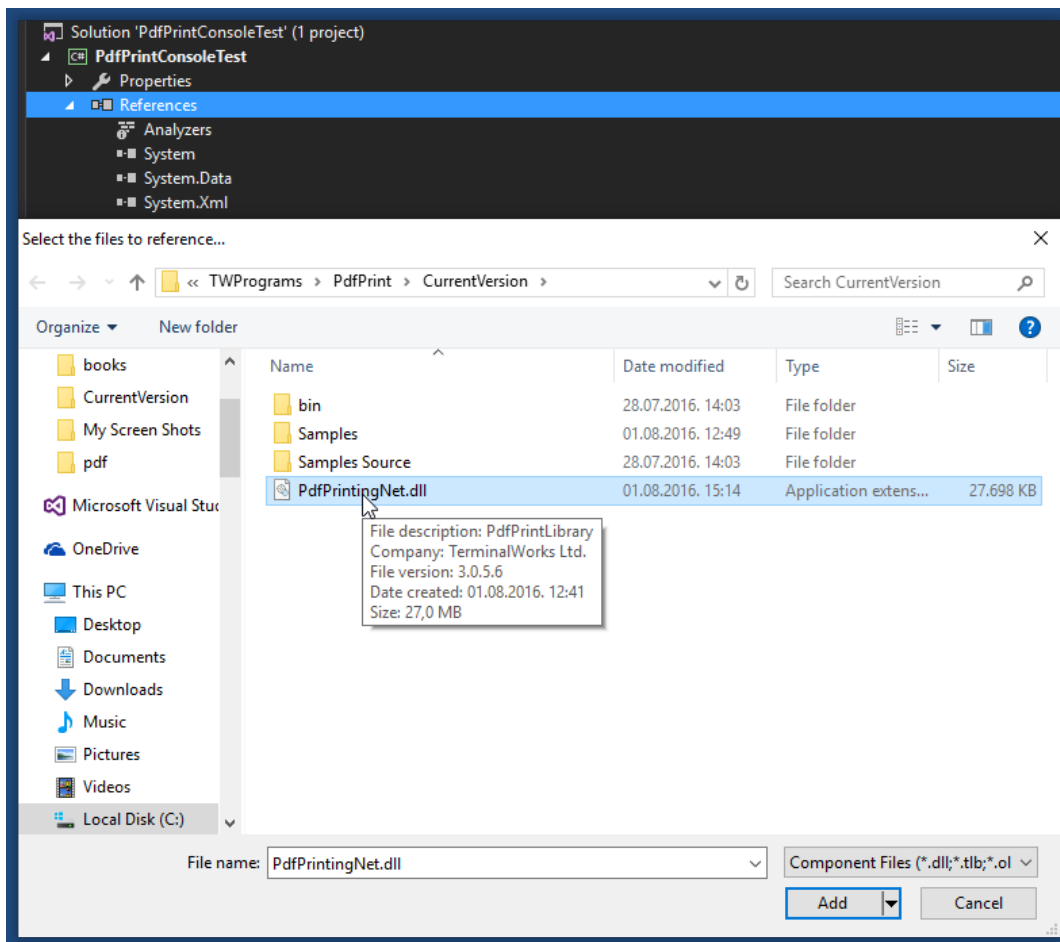
## Installation for print or convert to images functionality

1. Create a new project.

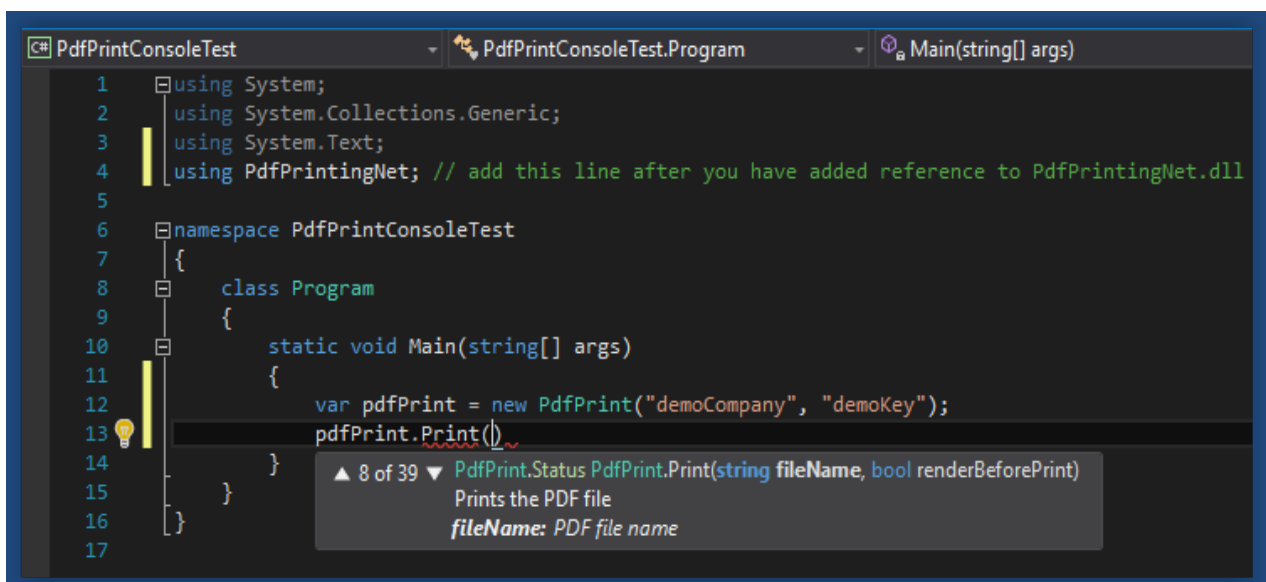


We are creating the console application, but it could also be used in WindowsForms, ASP.NET, WindowsServices or Class Library type of projects. Also, it will work in 32, 64 and AnyCPU bit applications.

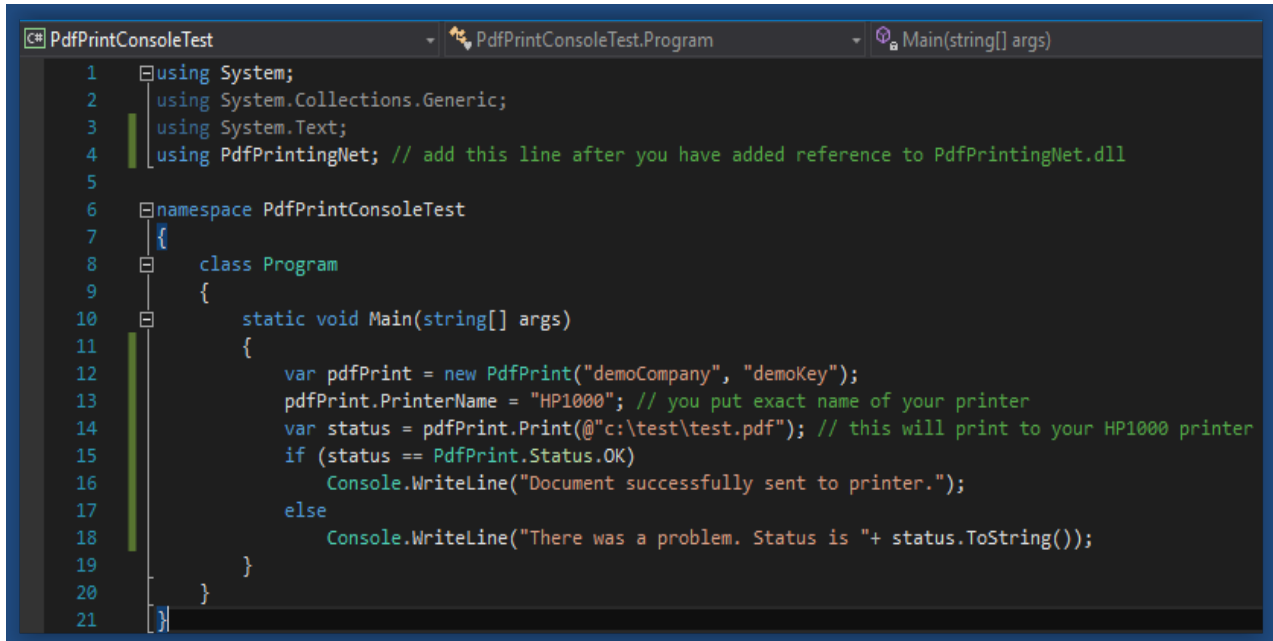
2. Extract the PdfPrinting.zip and add reference to the PDFPrining.dll



3. Start using our pdfprint library in your code

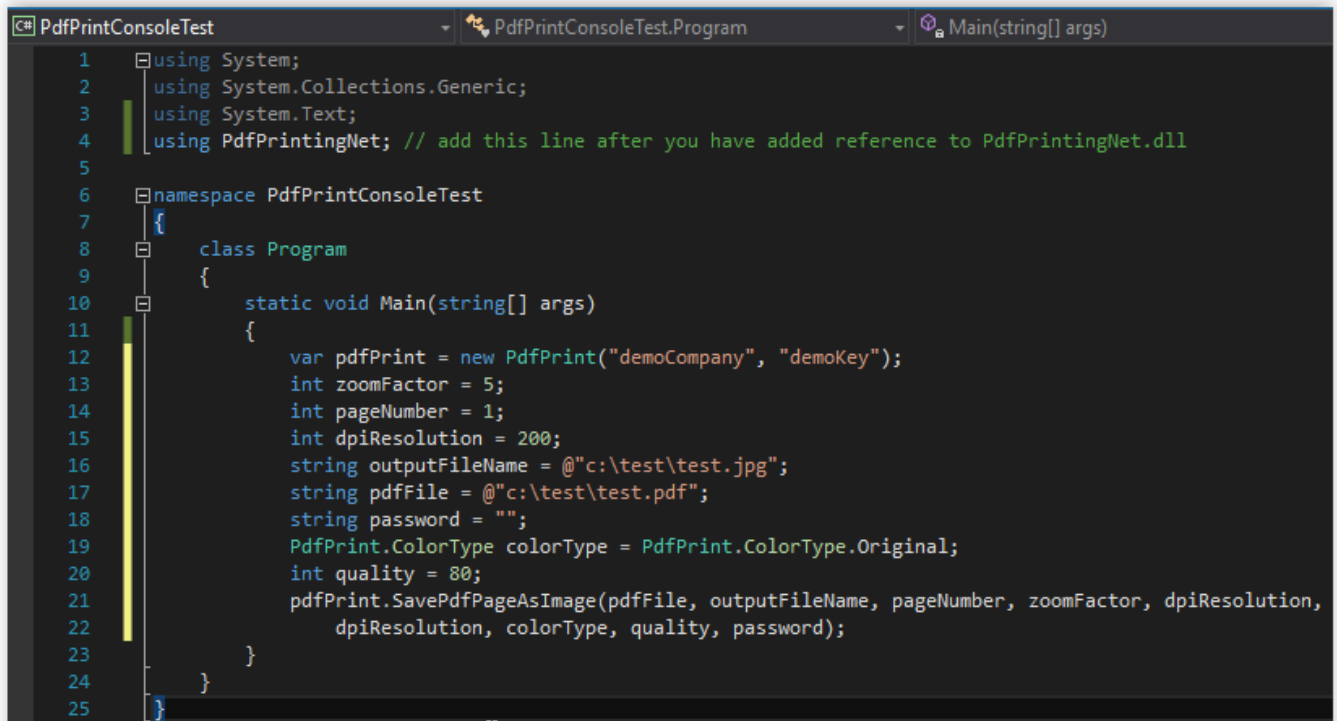


4. Finish this simple example and execute the code.



```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using PdfPrintingNet; // add this line after you have added reference to PdfPrintingNet.dll
5
6 namespace PdfPrintConsoleTest
7 {
8     class Program
9     {
10         static void Main(string[] args)
11         {
12             var pdfPrint = new PdfPrint("demoCompany", "demoKey");
13             pdfPrint.PrinterName = "HP1000"; // you put exact name of your printer
14             var status = pdfPrint.Print(@"c:\test\test.pdf"); // this will print to your HP1000 printer
15             if (status == PdfPrint.Status.OK)
16                 Console.WriteLine("Document successfully sent to printer.");
17             else
18                 Console.WriteLine("There was a problem. Status is " + status.ToString());
19         }
20     }
21 }
```

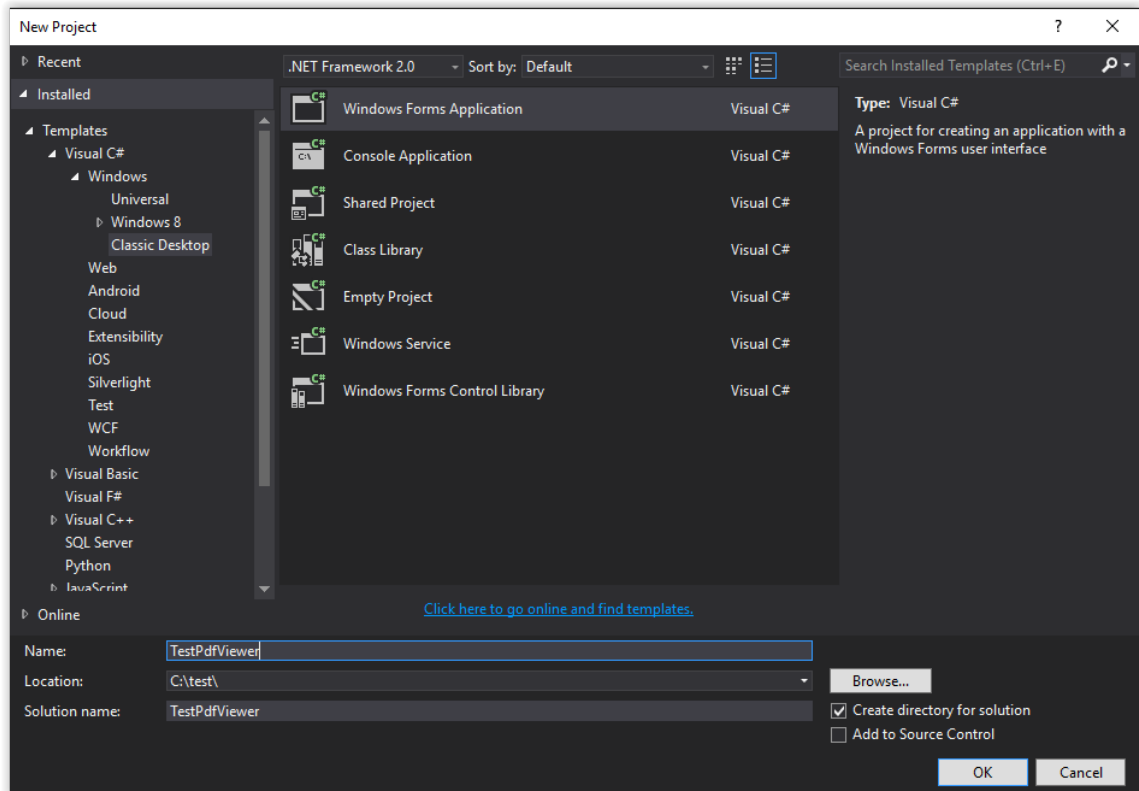
5. To test converting to image



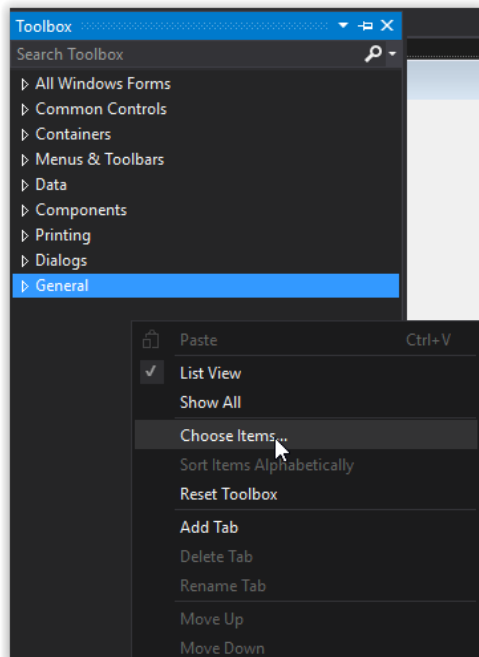
```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using PdfPrintingNet; // add this line after you have added reference to PdfPrintingNet.dll
5
6 namespace PdfPrintConsoleTest
7 {
8     class Program
9     {
10         static void Main(string[] args)
11         {
12             var pdfPrint = new PdfPrint("demoCompany", "demoKey");
13             int zoomFactor = 5;
14             int pageNumber = 1;
15             int dpiResolution = 200;
16             string outputFileName = @"c:\test\test.jpg";
17             string pdfFile = @"c:\test\test.pdf";
18             string password = "";
19             PdfPrint.ColorType colorType = PdfPrint.ColorType.Original;
20             int quality = 80;
21             pdfPrint.SavePdfPageAsImage(pdfFile, outputFileName, pageNumber, zoomFactor, dpiResolution,
22                                     dpiResolution, colorType, quality, password);
23         }
24     }
25 }
```

## Installation for PdfViewer functionality

1. Create new project – Windows Form Application

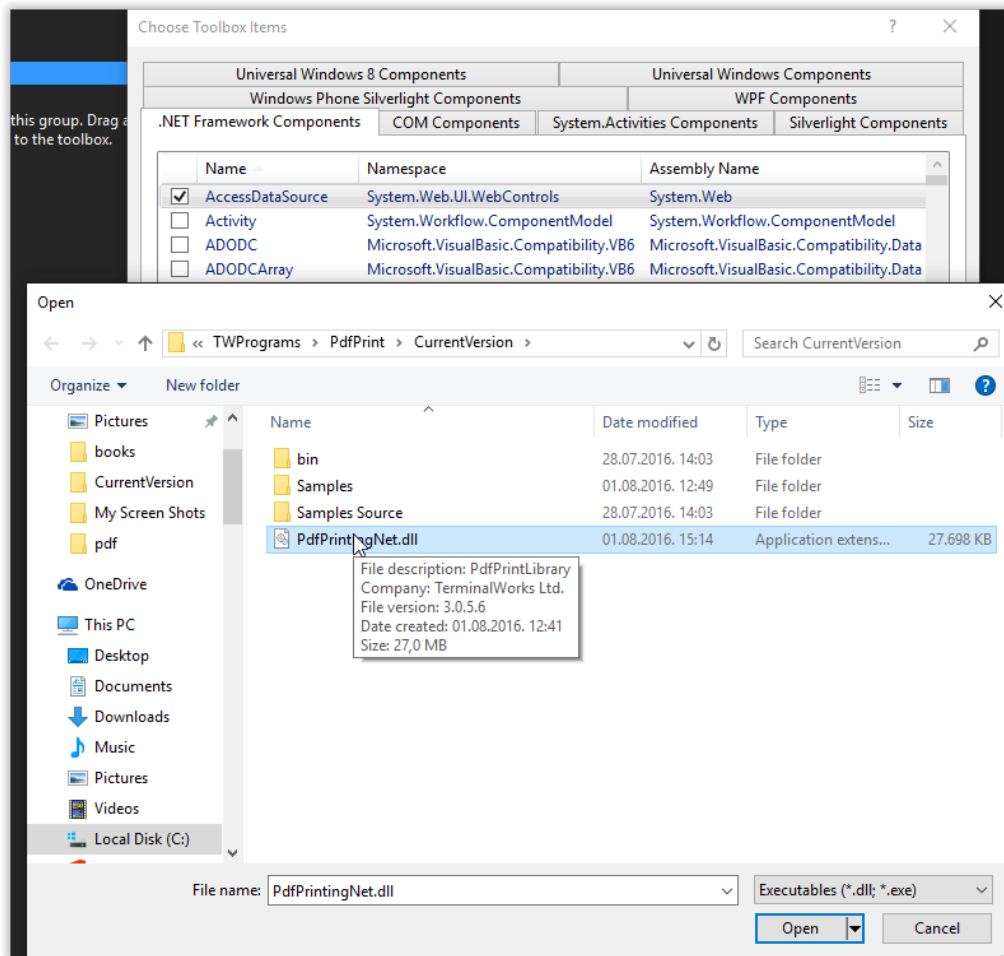


2. Add reference to PdfPrintingNet.dll
3. Build your project.
4. Add PdfViewer component to the Visual Studio toolbox. *Choose Items...* on toolbox

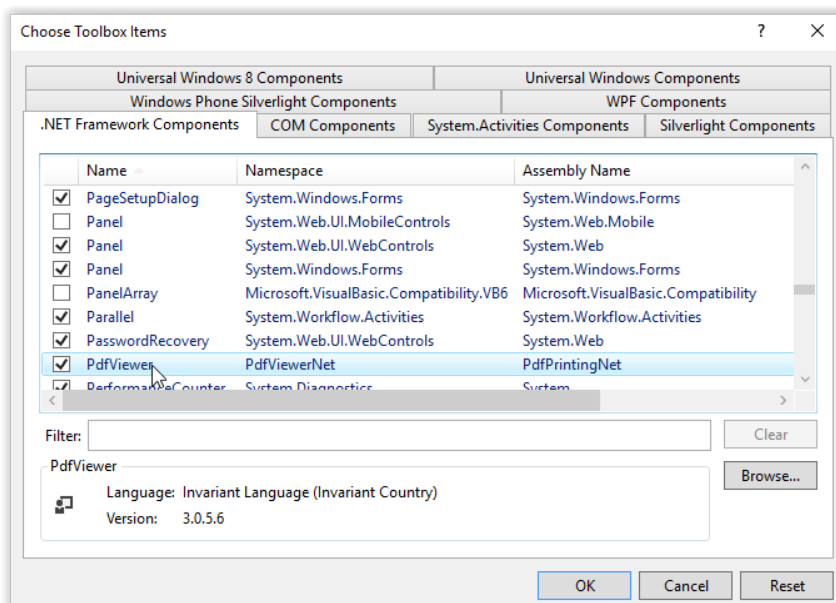




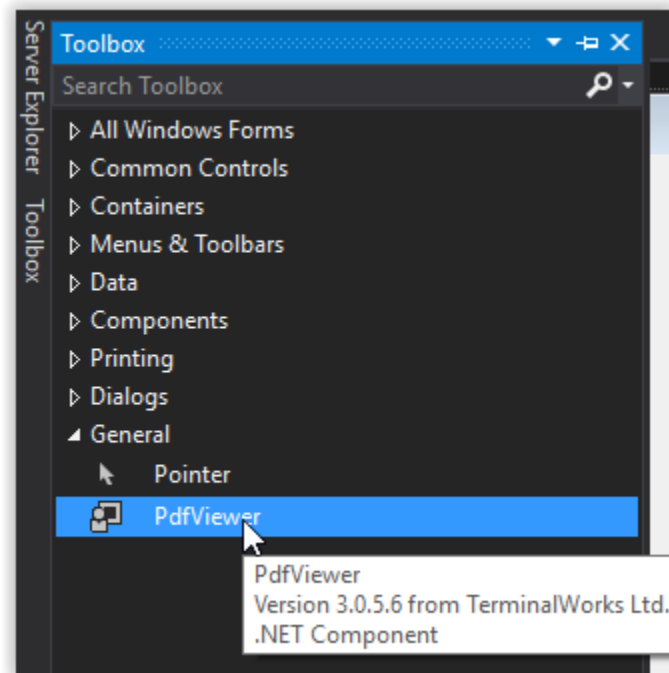
5. Browse and select PdfPrintingNet.dll



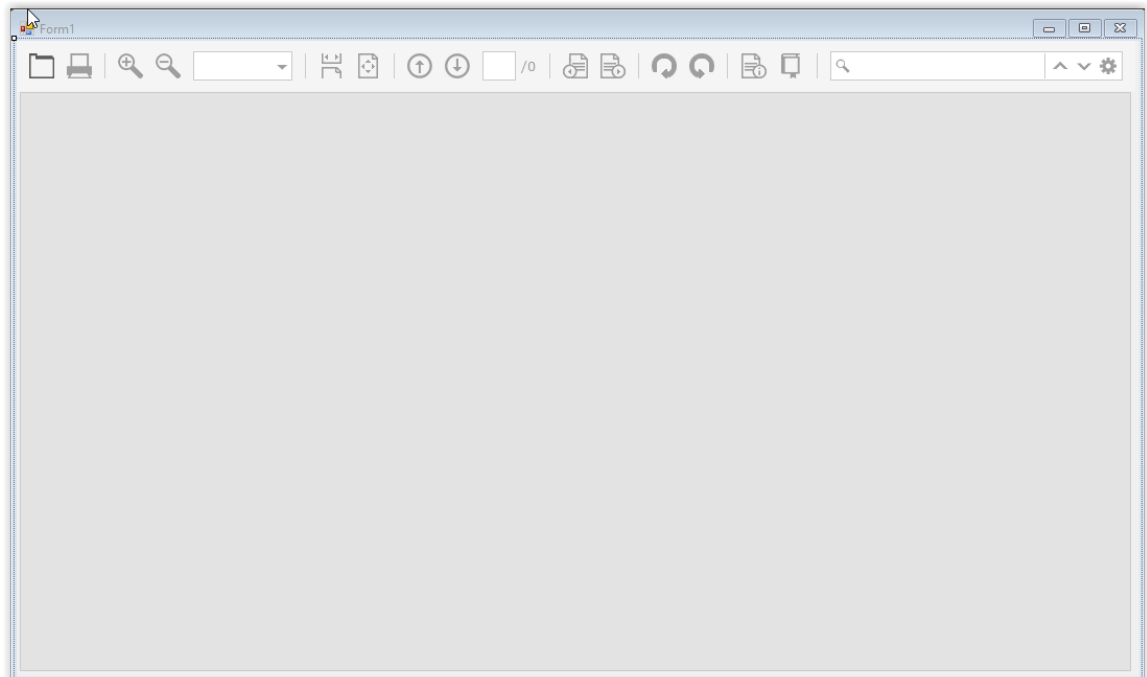
6. That will add PdfViewer component in .NET Framework Components tab. Select it and press OK.



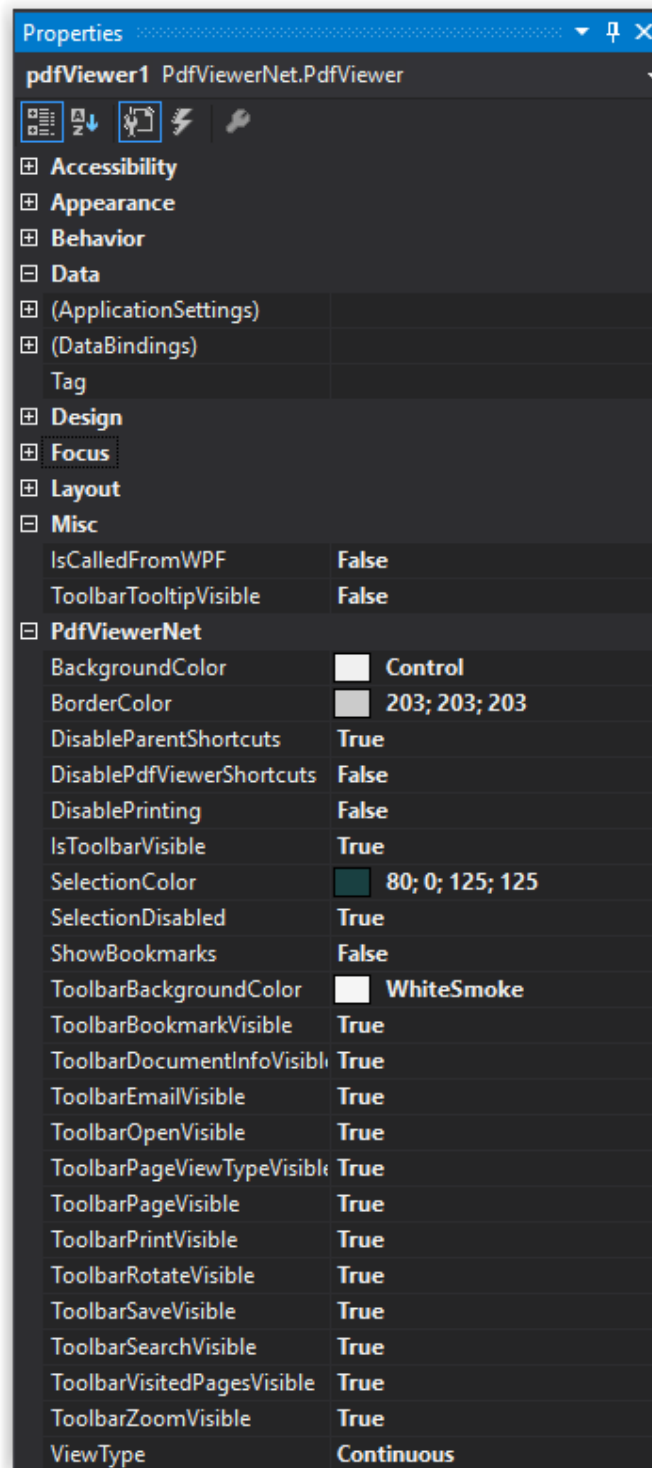
7. PdfViewer is now part of Visual Studio toolbox.



8. Drag PdfViewer control from Toolbox to your form, and you will get something like this:



9. In design time, from Visual Studio, you can change its properties. You can also change all properties from source code.



10. Run your demo app.

## Available Shortcuts in PdfViewer Component

Shortcut	Operation
CTRL + A	Select all text
CTRL + SHIFT + A	Deselect all text
CTRL + C	Copy selected text or image to clipboard
SHIFT + RIGHT	Go to next page
CTRL + P	Print
CTRL + O	Open
CTRL + F	Search
CTRL + Y	Zoom to
CTRL + 0	Zoom to fit to page
CTRL + 1	Zoom to actual size
CTRL+S	Save the opened document
CTRL + L	Email the opened document
CTRL + W	Close the document
CTRL + B	Toggle bookmarks
CTRL + D	Show document info
CTRL + SHIFT + ADD	Rotate clockwise
CTRL + SHIFT + SUBTRACT	Rotate counter-clockwise
PAGE UP	Go to the previous page
PAGE DOWN	Go to next page
SHIFT + LEFT	Go to the previous page
SHIFT + RIGHT	Go to next page
CTRL + SHIFT + N	Go to page
HOME	Go to the first page
END	Go to the last page
ALT + LEFT	Go to the previously visited page
ALT + RIGHT	Go to next visited page

## PdfPrint Examples

### *Simple silent printing of PDF files*

Silent printing enables you to print out documents in the background without user intervention. The below code sample will print out the PDF document to your default printer with default settings.

```
var pdfPrint = new PdfPrint("demoCompany", "demoKey");
string pdfFile = @"c:\test\test.pdf";
int numberOfPages = pdfPrint.GetNumberOfPages(pdfFile);
var status = pdfPrint.Print(pdfFile);
if (status == PdfPrint.Status.OK) { // check the result status of the Print method
    // your code here
}
// if you have pdf document in byte array that is also supported
byte[] pdfContent = YourCustomMethodWhichReturnsPdfDocumentAsByteArray();
status = pdfPrint.Print(pdfFile);
```

### *Silent printing with already installed Adobe Reader*

In case you would like to print using Adobe Reader for some specific document where our rendering doesn't meet your needs. The below sample code will invoke Adobe Reader in the background without being visible and print out the document.

```
var pdfPrint = new PdfPrint("demoCompany", "demoKey");
string pdfFile = @"c:\test\test.pdf";
var status = pdfPrint.PrintWithAdobe(pdfFile);
PrinterSettings printerSettings = YourCustomMethodWhichReturnsPrinterSettings();
status = pdfPrint.PrintWithAdobe(pdfFile, printerSettings);
```

### *Using PrinterSettings in Printing*

In case you decide to allow the end user to set the desired printer and print settings you can easily show a print dialog following the below sample.

```
var pdfPrint = new PdfPrint("demoCompany", "demoKey");
string pdfFile = @"c:\test\test.pdf";
PrinterSettings printerSettings = YourCustomMethodWhichReturnsPrinterSettings();
var status = pdfPrint.Print(pdfFile, printerSettings);
// if you have pdf document in byte array that is also supported
byte[] pdfContent = YourCustomMethodWhichReturnsPdfDocumentAsByteArray();
status = pdfPrint.Print(pdfContent, printerSettings);
// if you have pdf document in byte array that is also supported
pdfPrint.SettingDialog = true; // it will show printer dialog
status = pdfPrint.Print(pdfFile);
```

## Convert PDF documents into images

You can convert any PDF document to the following image formats: TIFF, JPG/JPEG, BMP, PNG, and GIF. You can convert just one page, the range of pages or all pages from the PDF document. You can convert a PDF document to a multi-page TIFF file.

```
var pdfPrint = new PdfPrint("demoCompany", "demoKey");
string pdfFile = @"c:\test\test.pdf";
int pageNumber = 2;
Bitmap bitmap = pdfPrint.GetBitmapFromPdfPage(pdfFile, pageNumber);
var status = pdfPrint.SavePdfPageAsImage(pdfFile, @"c:\test\test.jpg",
pdfPrint.GetNumberOfPages(pdfFile)); // save Last page as jpg
int fromPage = 3;
int toPage = 6;
status = pdfPrint.SavePdfPageAsImages(pdfFile, @"c:\test\newtest.jpg", fromPage, toPage);
status = pdfPrint.SavePdfPagesAsMultiPageTiff(pdfFile,
        @"c:\test\newtest.tiff", fromPage, toPage);
// if you have pdf document in byte array that is also supported
byte[] pdfContent = YourCustomMethodWhichReturnsPdfDocumentAsByteArray();
Bitmap bitmap2 = pdfPrint.GetBitmapFromPdfPage(pdfContent, pageNumber);
var status = pdfPrint.SavePdfPageAsImage(pdfContent, @"c:\test\test.jpg",
pdfPrint.GetNumberOfPages(pdfContent)); // save Last page as jpg
status = pdfPrint.SavePdfPageAsImages(pdfContent, @"c:\test\newtest.jpg", fromPage, toPage);
status = pdfPrint.SavePdfPagesAsMultiPageTiff(pdfContent,
        @"c:\test\newtest.tiff", fromPage, toPage);
```

## Working with password protected PDF documents

The following sample demonstrates how to check if the document is password protected. After that, it checks the password if it is valid and prints the document out.

```
var pdfPrint = new PdfPrint("demoCompany", "demoKey");
string pdfFile = @"c:\test\test.pdf";
if (!pdfPrint.IsSupportedFile(pdfFile)) ;
    return;//invalid pdf document
if (!pdfPrint.IsPasswordProtected(pdfFile))
    if (!pdfPrint.IsValidPassword(pdfFile, password))
        return;//invalid password
status = pdfPrint.Print(pdfFile, password);
// if you have pdf document in byte array that is also supported
byte[] pdfContent = YourCustomMethodWhichReturnsPdfDocumentAsByteArray();
if (!pdfPrint.IsSupportedFile(pdfContent))
    return;//invalid pdf document
if (!pdfPrint.IsValidPassword(pdfContent, password))
    return;//invalid password
status = pdfPrint.Print(pdfContent, password);
```

## Custom print properties

Please see an example how to set page range, landscape, copies, resolution, scaling, print in color, paper size, source tray, collate, printer name or duplex settings manually in your code.

```
var pdfPrint = new PdfPrint("demoCompany", "demoKey");
string pdfFile = @"c:\test\test.pdf";
pdfPrint.Collate = true;
pdfPrint.Copies = 2;
pdfPrint.DuplexType = System.Drawing.Printing.Duplex.Simplex;
pdfPrint.IsLandscape = true;
pdfPrint.PrintInColor = true;
pdfPrint.Pages = "2-6";
// pdfPrint.PaperSize = see our demo application source for example
// pdfPrint.PaperSource = see our demo application source for example
pdfPrint.PrinterName = "Your printer name";
// pdfPrint.PrinterResolution = see our demo application source for example
pdfPrint.RangeType = PdfPrint.RangeTypes.JustEven;
pdfPrint.Scale = PdfPrint.ScaleTypes.FitToMargins
var status = pdfPrint.Print(pdfFile);
```

## PdfViewer Examples

### *Open Document with open document status*

```
var pdfViewer = new PdfViewer();
pdfViewer.DocumentLoaded += PdfViewer_DocumentLoaded;
PdfOpenFileStatus status = pdfViewer.OpenDocument(@"c:\test\test.pdf");
string msg = "";
if (status.Result == PdfOpenFileStatus.PdfOpenFileResult.OK)
    msg = string.Format("Document has {0} pages. Current page is {1}. ",
pdfViewer.NumberOfPages, pdfViewer1.CurrentPageNumber);
else
    msg = "There was a problem loading document. Status = " + status.Result.ToString() + "
Additional information = " + status.Status;
MessageBox.Show(msg);

private void PdfViewer_DocumentLoaded(object sender, DocumentLoadedEventArgs e)
{
    MessageBox.Show("Document " + e.FileName + " was loaded");
}
```

### *Toolbar options – change different Toolbar properties*

```
var pdfViewer = new PdfViewer();
pdfViewer.IsToolbarVisible = true;
pdfViewer.ToolbarBackgroundColor = Color.LightBlue;
pdfViewer.ToolbarBookmarkVisible = true;
pdfViewer.ToolbarDocumentInfoVisible = false;
pdfViewer.ToolbarOpenVisible = true;
pdfViewer.ToolbarPageViewTypeVisible = true;
pdfViewer.ToolbarPageVisible = true;
pdfViewer.ToolbarPrintVisible = false;
pdfViewer.ToolbarRotateVisible = false;
pdfViewer.ToolbarSearchVisible = true;
pdfViewer.ToolbarVisitedPagesVisible = true;
pdfViewer.ToolbarZoomVisible = true;
```



## Print and its events

```
var pdfViewer = new PdfViewer();
pdfViewer.OpenDocument(@"c:\test\test.pdf");
pdfViewer.DocumentPrintBegin += PdfViewer_DocumentPrintBegin;
pdfViewer.DocumentPrintEnd += PdfViewer_DocumentPrintEnd;
pdfViewer.DocumentPrintPageBegin += PdfViewer_DocumentPrintPageBegin;
pdfViewer.DocumentPrintPageEnd += PdfViewer_DocumentPrintPageEnd;
PdfPrint.Status status = pdfViewer.Print();

private void PdfViewer_DocumentPrintPageBegin(object sender,
PdfPrint.DocumentPrintPageEventArgs e)
{
    string msg = "Print page begin. CurrentPageIndex=" + e.CurrentPageIndex + " Page Number=" +
        e.PageNumber + " Total Pages to Print=" + e.TotalNumberOfPagesToPrint;
}

private void PdfViewer_DocumentPrintPageEnd(object sender,
PdfPrint.DocumentPrintPageEventArgs e)
{
    string msg = "Print page end. CurrentPageIndex=" + e.CurrentPageIndex + " Page Number=" +
        e.PageNumber + " Total Pages to Print=" + e.TotalNumberOfPagesToPrint;
}

private void PdfViewer_DocumentPrintEnd(object sender, PdfPrint.DocumentPrintEndEventArgs e)
{
    string msg = "Print document end. Document = " + e.FileName + "Status=" + e.Status +
        " Number of pages in document=" + e.NumberOfPagesInDocument +
        " Number of pages printed:" + e.NumberOfPagesPrinted;
}

private void PdfViewer_DocumentPrintBegin(object sender, PdfPrint.DocumentPrintBeginEventArgs e)
{
    string msg = "Print document end. Document = " + e.FileName +
        " Number of pages in document=" + e.NumberOfPagesInDocument +
        " Number of pages to print=" + e.NumberOfPagesToPrint;
}
```

## Zoom functionality

```
var pdfViewer = new PdfViewer();
pdfViewer.OpenDocument(@"c:\test\test.pdf");
pdfViewer.ZoomChanged += PdfViewer_ZoomChanged;
pdfViewer.ZoomIn();
pdfViewer.ZoomOut();
float maxZoomValue = pdfViewer.GetMaxZoomLevel();
float minZoomValue = pdfViewer.GetMinZoomLevel();
List<float> availableZoomValues = pdfViewer.GetAvailableZoomLevels();
float currentZoomValue = pdfViewer.ZoomValue;
pdfViewer.ZoomValue = 2f; // This is equivalent to 200%

private void PdfViewer_ZoomChanged(object sender, ZoomEventArgs e)
{
    MessageBox.Show("Zoom value changed. New value is " + e.ZoomLevel);
}
```

## PDF page navigation

```
var pdfViewer = new PdfViewer();
pdfViewer.OpenDocument(@"c:\test\test.pdf");
pdfViewer.CurrentPageChanged += PdfViewer_CurrentPageChanged;
pdfViewer.GoToLastPage();
pdfViewer.GoToFirstPage();
pdfViewer.GoToNextPage();
pdfViewer.GoToPreviousVisitedPage();
pdfViewer.GoToPage(pdfViewer.NumberOfPages); // go to last page
pdfViewer.GoToPreviousPage();
pdfViewer.GoToNextVisitedPage();
// This are just getters
int currentPage = pdfViewer.CurrentPageNumber;
int currentVisitedPageIndex = pdfViewer.CurrentVisitedPageIndex;
List<int> visitedPages = pdfViewer.VisitedPages;

private void PdfViewer_CurrentPageChanged(object sender, CurrentPageEventArgs e)
{
    MessageBox.Show("Current page now is " + e.CurrentPage);
}
```

## Selection

```
var pdfViewer = new PdfViewer();
pdfViewer.OpenDocument(@"c:\test\test.pdf");
pdfViewer.SelectionChanged += PdfViewer_SelectionChanged;
pdfViewer.SelectionColor = Color.LightGreen;
pdfViewer.SelectAllText(1); // it will select all text on first page
string textOnFirstPage = pdfViewer.GetSelectedText();
Image image = pdfViewer.GetSelectedImage();
pdfViewer.SelectionDisabled = true; // selection will not be possible

private void PdfViewer_SelectionChanged(object sender, SelectionEventArgs e)
{
    MessageBox.Show("Selection Type is " + e.SelectionType + " and number of selected items is " +
        e.SelectedCount);
}
```

## Miscellaneous properties and functions

```
var pdfViewer = new PdfViewer();
pdfViewer.SetLicenseInfo("your company name", "your license key");
pdfViewer.OpenDocument(@"c:\test\test.pdf");
// set focus on appropriate toolbar field
pdfViewer.SelectToolbarPageNumberField();
pdfViewer.SelectToolbarSearchField();
pdfViewer.SelectToolbarZoomField();
pdfViewer.Rotate(true); // rotates clockwise

pdfViewer.ShowBookmarks = true;
pdfViewer.ShowDocumentInfo();
DocumentPermissions documentPermissions = pdfViewer.DocumentPermissions;
```

## Search

```
var pdfViewer = new PdfViewer();
pdfViewer.OpenDocument(@"c:\test\test.pdf");
string searchTerm = "test";
bool isCaseSensitive = true;
bool wholeWordOnly = true;
// it will select next occurrence of testWord
pdfViewer.SearchNext(searchTerm, isCaseSensitive, wholeWordOnly);
// it will select previous occurrence of testWord
pdfViewer.SearchPrevious(searchTerm, isCaseSensitive, wholeWordOnly);
```

## PdfPrint FAQ

### *Where can I see a code sample for usage of the PDFPrinting.NET?*

Code samples with Visual Studio solution file you can find in PdfPrinting.zip located in <http://www.terminalworks.com/pdf-print-net/downloads>

There is a code of our PdfPrintLibraryTest demo application.

In that code, you can notice:

- How to set all available properties in PdfPrint library (PaperSource, Collate, Copies, Pages, Duplex, Print in Color, Print range, Scale document, Landscape / Portrait, Printer Resolution, Paper Size, Show Printer dialog)
- How to change default printer in C# and get default values for default printer
- How to get a list of available printers

### *Can I use PDFPrinting.NET in windows service?*

Yes, the most important thing to know is that user running Windows service where pdfprint is used must have rights to print to selected printer - otherwise, it will not work.

Set up windows service to be run as a user account type.

User account type - causes the system to prompt for a valid username and password when the service is installed and runs in the context of an account specified by a single user on the network. If you want to start windows service as local system account, then system local account must have printer rights.

**Note:** Following instruction were found at <http://support.microsoft.com/kb/184291>

To set up printers for the SYSTEM account, perform the following:

This method requires you to modify the registry using the Registry Editor.

**Warning:** Using Registry Editor incorrectly can cause serious, system-wide problems that may require you to reinstall Windows to correct them.

Microsoft cannot guarantee that any problems resulting from the use of Registry Editor can be solved.

Use this tool at your risk.

Ensure that the user you are currently logged into on the server has the desired printers installed.

Launch the Registry Editor (Regedit.exe).

Select the following key:

***HKEY\_CURRENT\_USER\Software\Microsoft\Windows NT\Current Version\Devices***

From the Registry menu, click Export Registry File.

In the File Name text box, type *c:\Devices.reg*.

Select the following key:

***HKEY\_CURRENT\_USER\Software\Microsoft\Windows NT\Current Version\PrinterPorts***

From the Registry menu, click Export Registry File.

In the File Name text box, type *c:\PrinterPorts.reg*.

Select the following key:

***HKEY\_CURRENT\_USER\Software\Microsoft\Windows NT\Current Version\Windows***

From the Registry menu, click Export Registry File.

In the File Name text box, type *c:\Windows.reg*.

From the Start button, select Run. Open *Devices.reg* in Notepad by typing Notepad *Devices.reg* in Run dialog box.

Replace the text ***HKEY\_CURRENT\_USER*** with ***HKEY\_USERS\DEFAULT***

Save the file. Then import it into the registry by double-clicking the file in Windows Explorer.

Repeat steps are done for *Devices.reg* for *PrinterPorts.reg* and *Windows.reg*.

These steps only work for local printers.

If you are using *PrintWithAdobe* method, please also read *Can PdfPrint use Adobe for silent printing?*

### ***Can I use PDFPrinting.NET to print two pages to one page/ merge two pdf documents?***

Unfortunately, we don't support that option since we are a printing library.

### ***Can PDFPrinting.NET use Adobe for silent printing?***

*PrintWithAdobe* method uses preinstalled Adobe Reader / Professional for silent printing.

It works for 32 and 64-bit processes. That method will not work in this cases:

- in windows service / ASP.net service it will not work if installed version of Adobe Reader is higher than 9.5
- Adobe Reader / Professional isn't already installed on computer where pdfprint library is used

Printing with default pdfprint engine doesn't have above limitations.

### ***Can I use PDFPrinting.NET to print to file?***

Unfortunately, we don't support that option.

## *Are paper sources (input trays) and output trays supported?*

**PdfPrint library has support for paper sources.**

```
private static PaperSource GetPaperSource(PrinterSettings printerSettings, string paperSourceName)
{
    foreach (PaperSource paperSource in printerSettings.PaperSources)
    {
        if (paperSource.ToString().Equals(paperSourceName))
            return paperSource;
    }
    return null; // Paper source not found
}

var pdfprint = new PdfPrint("your company name", "your key");
var printerSettings = new PrinterSettings();
pdfprint.PaperSource = GetPaperSource("name of your input tray"); // it is case sensitive
```

**PdfPrint library doesn't have support for output trays.**

.NET PrinterSettings class has property for setting PaperSource. PaperSource is standard printer setting.

Output trays aren't standard printer property - and every printer which supports it handle it differently, so it isn't possible to have one general solution which will work for every printer.

So what to do if you want to print to specific output tray?

The only option is that you use property SettingDialog to show Printer settings dialog.

If not set through SettingDialog, printer default output tray will be used.

## *Can I include PdfPrintingNet.dll in my setup file? (ClickOnce, MSI, ...)*

PdfPrintingNet.dll is a standalone dll library. It is signed with a public key.

It could be included in ClickOnce, MSI and other types of Windows setup file.

## *How page scaling in printing works?*

PdfPrint has three different scaling options in printing:

1. **Actual Size** - it leaves the page as it is. If the content of the page is larger than printer printable area than part of the content will be truncated.
2. **Fit to margin** - if the content of the page is bigger or smaller than the printable area, content will be resized, so it fits printer printable area.
3. **Shrink to margins** - if the content of the page is larger than printable area, content will be downsized. If it isn't bigger than printable area, content size will stay the same.

Different printers have different printable area.

## *Is PdfPrintingNet.dll signed with public key?*

PdfPrintingNet.dll is signed with the public key.

## *Can I merge pdfprint dll with ilmerge tool?*

Unfortunately, it will not work.

Executing ilmerge to merge pdfprint dll with another executable will not give you any error, but still, it won't work.

PdfPrint dll is standalone and must stay like that - located in the same folder as the executable which uses pdf print library.

```
var pdfPrint = new PdfPrint("demoCompany", "demoKey");
string pdfFile = @"c:\test\test.pdf";
int numberOfPages = pdfPrint.GetNumberOfPages(pdfFile);
var status = pdfPrint.Print(pdfFile);
if (status == PdfPrint.Status.OK) { // check the result status of the Print method
    // your code here
}
// if you have pdf document in byte array that is also supported
byte[] pdfContent = YourCustomMethodWhichReturnsPdfDocumentAsByteArray();
status = pdfPrint.Print(pdfFile);
```

## *Can I convert images to pdf?*

No, the only the pdf to the image is possible.

## *How can I license the software? What is needed in my redistributable application?*

When you buy the license, you will receive serial key valid only for your company.  
Instead of:

```
var pdfPrint = new PdfPrint ("demoCompany", "demoKey");
```

you will have in your code:

```
var pdfPrint = new PdfPrint ("your company name", "your license key");
```

There isn't any need for additional license files.

## *What are limitations of the demo version of pdfprint library?*

When used with the method `PrintWithAdobe` there is 20 seconds delay before printing with the popup dialog. You can close manually popup dialog after 5 seconds.

When converting PDF pages to images, there is a watermark on the created image.

There aren't any additional limitations.

## *What is the difference between Viewer, Print, and Full license?*

**Viewer** license allows displaying PDF document in Viewer component without any limitations. Also, it allows printing loaded document directly from Viewer component. Viewer license doesn't support silent printing.

**Print** license allows to print or convert PDF document silently without any limitation. If the Viewer component displays the document, it will show demo sign. If the Viewer component prints the document, then it will print it without demo sign.

**Full** license allows to print, convert and display PDF document without any demo limitation.



## PdfViewer component FAQ

### *How to license PdfViewer object?*

First, you need valid Viewer or Full license.

If your license type is Viewer or Full than you have valid Viewer license and you have received the correct license key in an email when you purchase PdfPrintingNet library.

In your source code, you will have something like this:

```
var pdfViewer = new PdfViewer();  
pdfViewer.SetLicenseInfo("your company name", "your license key");
```

### *When calling the Print() method on the Viewer component I get the print dialog. Can I silently print a document?*

No, for advanced print settings or silent printing you need to use the Full license of our PDFPrinting.NET library.

### *Can I remove some of the options on the toolbar of Viewer component?*

Yes, you can hide toolbar completely, or you can hide just some of the options of the toolbar. For example, you could hide the page selection elements and then only allow your users to view a single page.

You can change it:

1. from Visual Studio in design time - in properties window of the toolbar
2. programmatically from you code

### *Why is a print button on the Viewer component grayed out?*

That occurs in case the PDF document which you opened has permissions set to disallow printing. You can check out the permission of the document by calling the GetPermissions method on the component.

### *How to add PdfViewer component on my Windows Form in design time from Visual Studio?*

1. In your project reference PdfPrintingNet.dll
2. Build your project.
3. In Visual Studio go to Toolbox -> Choose Items -> Browse... -> Select PdfPrintingNet.dll on your disk
4. That will add PdfViewer in .NET framework components. Check it and press OK.

### ***Does Viewer component respects PDF permissions?***

Our library retrieves following permissions from PDF document:

- Print
- Modify
- Comments / Annotations
- Document Assembly
- Content Extraction
- Content Extraction For Accessibility
- Filling of form fields
- Full qualify print

The current version of our library uses only Print / Full quality print and Content Extraction / Content Extraction For Accessibility. Other are available only as information.

In the future version where it will be possible to edit PDF document, other permissions will be respected.

### ***What are limitations of the demo version of PdfViewer?***

At the bottom of the displayed PDF document, there is small demo watermark.

If the document is printed from the Viewer component, on the printed pages there is demo watermark.

There isn't any other limitation.

### ***Can I use PdfViewer in WPF application?***

Yes, it is possible to use it through WindowsFormHost component.

We recommend that you look at our demo application PdfViewerPDFDemo and its source located in PdfPrinting.zip.