User Manual

3-Heights®
PDF Printer API

Version 6.27.10

pdftools

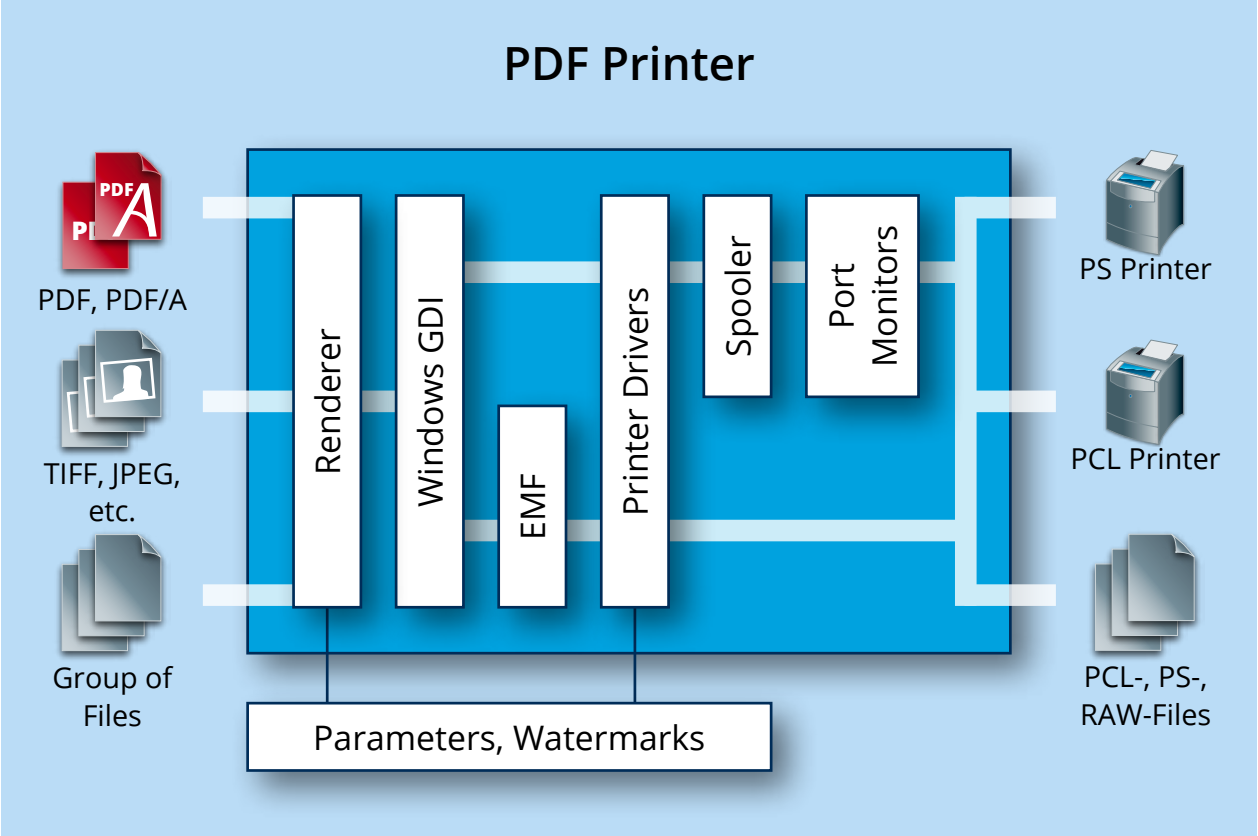
Contents

1 Introduction

1.1 Description

The 3-Heights® PDF Printer API is an efficient and practical solution for automated (background) printing of PDF documents on all Windows printers including PostScript and PCL, and on virtual printers. A variety of options are available for printer control.

The tool is characterized first and foremost by its high level of performance and is extremely adaptable to specific requirements. It also supports PDF/A conformance printing.



1.2 Functions

The 3-Heights® PDF Printer API translates PDF, PDF/A, TIFF, and JPEG into printer driver language such as PostScript or PCL. Documents are either printed on a physical printer (local, remote, or via Internet) or issued as a file. The tool offers a variety of printer control options such as paper tray, paper format, duplex printing, stapling, merging multiple pages to form a single print job, and applying watermarks in the form of text and images. It is also possible to query the properties of the target printer (print margins, resolution, etc.) and to optimize printing accordingly. In addition to all current printer models, the tool supports older printers via emulation. The printer supports Citrix virtual printer drivers.

1.2.1 Features

- Print on paper or virtual printers and divert printing to a file
- Perform local / remote printing

- Select paper format
- Select paper tray
- Select print quality
- Define page sequence
- Select printer-specific properties
- Control color management
- Supports HTTP, HTTPS, and FTP data streams
- Print raster images (TIFF, JPEG, PNG, etc.)
- Group documents in one print job
- Integrate watermarks (text, image)
- List printers per host and printer properties (supported paper formats, trays, etc.)
- Perform duplex printing
- Select orientation
- Print multiple copies
- Determine positioning (centering, scaling, realigning)
- Print encrypted documents

1.2.2 Formats

Input formats:

- PDF 1.x (PDF 1.0, …, PDF 1.7)
- PDF 2.0
- PDF/A-1, PDF/A-2, PDF/A-3
- BMP
- GIF
- JBIG2
- JPEG
- JPEG2000, JPEG-LS
- PBM
- PNG
- TIFF

Output formats:

- Print spool formats such as PostScript, PCL 5, PCL 6, AFP

1.2.3 Conformance

Standards:

- ISO 32000-1 (PDF 1.7)
- ISO 32000-2 (PDF 2.0)
- ISO 19005-1 (PDF/A-1)
- ISO 19005-2 (PDF/A-2)
- ISO 19005-3 (PDF/A-3)
- TIFF V6

1.3 Interfaces

The following interfaces are available:

- C
- Java
- .NET Framework
- .NET Core[1]
- COM

1.4 Operating systems

The 3-Heights® PDF Printer API is available for the following operating systems:

- Windows Client 7+ | x86 and x64
- Windows Server 2008, 2008 R2, 2012, 2012 R2, 2016, 2019, 2022 | x86 and x64

'+' indicates the minimum supported version.

1.5 Compatibility note

Versions prior to 2010 distinguished between the Standard and the Pro version. Certain features such as watermarking and printing raster images were only supported in the Pro version. At present, there is only one version, which supports all features.

1.6 How to best read this manual

If you are reading this manual for the first time and would like to evaluate the software, the following steps are suggested:

1. Read the Introduction chapter to verify this product meets your requirements.
2. Identify what interface your programming language uses.
3. Read and follow the instructions in Installation and deployment.
4. In Programming interfaces, find your programming language. Please note that not every language is covered in this manual.
   For most programming languages, there is sample code available. To start, it is generally best to refer to these samples rather than writing code from scratch.
5. (Optional) Read the User guide for general information about the API. Read the Interface reference for specific information about the functions of the API.

[1] Limited supported OS versions. Operating systems

2 Installation and deployment

2.1 Windows

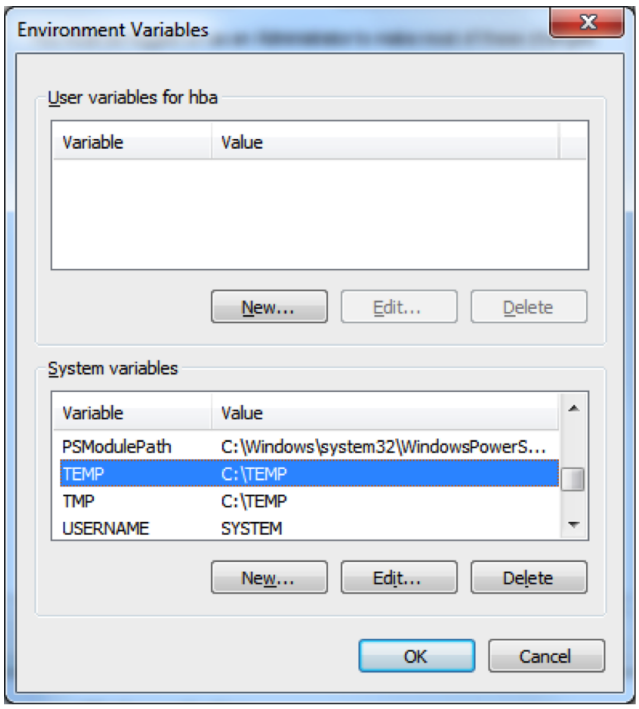
The 3-Heights® PDF Printer API comes as a ZIP archive or as a NuGet package.

To install the software, proceed as follows:

1. You need administrator rights to install this software.
2. Log in to your download account at https://www.pdf-tools.com. Select the product "PDF Printer API". If you have no active downloads available or cannot log in, please contact pdfsales@pdf-tools.com for assistance.
   You can find different versions of the product available. Download the version that is selected by default. You can select a different version.
   The product comes as a ZIP archive containing all files, or as a NuGet package containing all files for development in .NET.
   There is a 32 and a 64-bit version of the product available. While the 32-bit version runs on both 32 and 64-bit platforms, the 64-bit version runs on 64-bit platforms only. The ZIP archive as well as the NuGet package contain both the 32-bit and the 64-bit version of the product.
   It is strongly recommended that you use the 64-bit version on a 64-bit platform to avoid problems associated with the thunking process that translates between the 32-bit application and the 64-bit printer drivers.
3. If you are using the ZIP archive, uU nzip the archive to a local folder, e.g. `C:\Program Files\PDF Tools AG\`.
   This creates the following subdirectories (see also ZIP archive):

Subdirectory	Description
`bin`	Runtime executable binaries
`doc`	Documentation
`include`	Header files to include in your C/C++ project
`jar`	Java archive files for Java components
`lib`	Object file library to include in your C/C++ project
`samples`	Sample programs in various programming languages

4. The usage of the NuGet package is described in section NuGet package.
5. (Optional) Register your license key using the License management.
6. Identify the interface you are using. Perform the specific installation steps for that interface described in Interface-specific installation steps.
7. Ensure the system environment variable TMP exists and points to an existing directory. This directory is required to temporarily install fonts that are embedded in PDF documents.
   Control Panel → System → Advanced → Environment Variables



8. Ensure the cache directory exists as described in Special directories.
9. Make sure your platform meets the requirements regarding color spaces and fonts described in Color profiles and Fonts, respectively.

2.2 ZIP archive

The 3-Heights® PDF Printer API provides four different interfaces. The installation and deployment of the software depend on the interface you are using. The table below shows the supported interfaces and some of the programming languages that can be used.

Interface	Programming languages
.NET	The MS software platform .NET can be used with any .NET capable programming language such as: ■ C# ■ VB .NET ■ J# ■ others For a convenient way to use this interface, see NuGet package.
Java	The Java interface.

COM	The component object model (COM) interface can be used with any COM-capable programming language, such as: ■ MS Visual Basic ■ MS Office Products such as Access or Excel (VBA) ■ C++ ■ VBScript ■ others This interface is available in the Windows version only.
C	The native C interface is for use with C and C++.

2.2.1 Development

The software development kit (SDK) contains all files that are used for developing the software. The role of each file in each of the four different interfaces is shown in table Files for development. The files are split in four categories:

Req. The file is required for this interface.

Opt. The file is optional. See also the File description table to identify the files are required for your application.

Doc. The file is for documentation only.

Empty field An empty field indicates this file is not used for this particular interface.

Files for development

Name	.NET	Java	COM	C
bin\‹platform›\PdfPrintAPI.dll	Req.	Req.	Req.	Req.
bin\‹platform›\INET.dll	Opt.	Opt.	Opt.	Opt.
bin*NET.dll	Req.			
bin*NET.xml	Doc.			
doc*.pdf	Doc.	Doc.	Doc.	Doc.
doc\PrinterOCX.idl			Doc.	
doc\javadoc*.*		Doc.		
include\printer_c.h				Req.
include*.*				Opt.
jar\PRNA.jar		Req.		
lib\‹platform›\PdfPrintAPI.lib				Req.
samples*.*	Doc.	Doc.	Doc.	Doc.

The purpose of the most important distributed files is described in the File description table.

File description

Name	Description
bin\‹platform›\PdfPrintAPI.dll	DLL that contains the main functionality (required), where ‹platform› is either Win32 or x64 for the 23-bit or the 64-bit library, respectively. (Compatibility-Note: In previous versions, this DLL was called PrinterOCX.dll or PrinterProOCX.dll.)
bin\‹platform›\INET.dll	DLL that implements https: and ftp: connections using Internet Explorer. It is loaded from module path.
bin*NET.dll	.NET assemblies are required when using the .NET interface. The files bin*NET.xml contain the corresponding XML documentation for MS Visual Studio.
doc*.*	Documentation
include*.*	Files to include in your C / C++ project
lib\‹platform›\PdfPrintAPI.lib	The object file library needs to be linked to the C/C++ project.
jar\PRNA.jar	Java API archive
samples*.*	Sample programs in different programming languages

2.2.2 Deployment

For the deployment of the software, only a subset of the files are required. The table below shows the files that are required (Req.), optional (Opt.) or not used (empty field) for the four different interfaces.

Files for deployment

Name	.NET	Java	COM	C
bin\‹platform›\PdfPrintAPI.dll	Req.	Req.	Req.	Req.
bin\‹platform›\INET.dll	Opt.	Opt.	Opt.	Opt.
bin*NET.dll	Req.			
jar\PRNA.jar		Req.		

The deployment of an application works as described below:

1. Identify the required files from your developed application (this may also include color profiles).
2. Identify all files that are required by your developed application.
3. Include all these files in an installation routine such as an MSI file or a simple batch script.
4. Perform any interface-specific actions (e.g. registering when using the COM interface).

[2] These files must reside in the same directory as PdfPrintAPI.dll.

Example: This is a very simple example of how a COM application written in Visual Basic 6 could be deployed.

1. The developed and compiled application consists of the file `print.exe`. Color profiles are not used.
2. The application uses the COM interface and is distributed on Windows only.
   - The main DLL `PdfPrintAPI.dll` must be distributed.
   - The application supports HTTPS connections, therefore `Inet.dll` is distributed.
   - All documents used by the application have the corresponding fonts embedded (e.g. because they conform to PDF/A), therefore the font-related files are not distributed.
3. All files are copied to the target location using a batch script. This script contains the following commands:

```
copy print.exe %targetlocation%\.
copy PdfPrintAPI.dll Inet.dll %targetlocation%\.
```

4. For COM, the main DLL needs to be registered in silent mode (`/s`) on the target system. This step requires PowerUser privileges and is added to the batch script.

```
regsvr32 /s %targetlocation%\PdfPrintAPI.dll.
```

2.3 NuGet package

NuGet is a package manager that lets you integrate libraries for software development in .NET. The NuGet package for the 3-Heights® PDF Printer API contains all the libraries needed, both managed and native.

Installation

The package `PdfTools.PdfPrint 6.27.10` is available on nuget.org. Right-click on your .NET project in Visual Studio and select "Manage NuGet Packages...". Finally, select the package source "nuget.org" and navigate to the package `PdfTools.PdfPrint 6.27.10`.

Development

The package `PdfTools.PdfPrint 6.27.10` contains .NET libraries with versions .NET Standard 1.1, .NET Standard 2.0, and .NET Framework 2.0, and native libraries for Windows.

The required native libraries are loaded automatically. All project platforms are supported, including "AnyCPU".

To use the software, you must first install a license key for the 3-Heights® PDF Printer API. To do this, you have to download the product kit and use the license manager in it. See also License management.

> **Note:** This NuGet package is only supported on a subset of the operating systems supported by .NET Core. See also Operating systems.

2.4 Interface-specific installation steps

2.4.1 COM interface

Registration

Before you can use the 3-Heights® PDF Printer API component in your COM application program, you have to register the component using the `regsvr32.exe` program that is provided with the Windows operating system. The

following command shows how to register the PdfPrintAPI.dll. In Windows Vista and later, the command needs to be executed from an administrator shell.

```
regsvr32 "C:\Program Files\PDF Tools AG\bin\‹platform›\PdfPrintAPI.dll"
```

Where ‹platform› is Win32 for the 32-bit and x64 for the 64-bit version.

If you are using a 64-bit operating system and would like to register the 32-bit version of the 3-Heights® PDF Printer API, you need to use the regsvr32 from the directory %SystemRoot%\SysWOW64 instead of %SystemRoot%\System32.[3]

If the registration process succeeds, a corresponding dialog window is displayed. The registration can also be done silently (e.g. for deployment) using the switch /s.

Other files

The other DLLs do not need to be registered, but for simplicity, it is suggested that they reside in the same directory as the PdfPrintAPI.dll.

2.4.2 Java interface

The 3-Heights® PDF Printer API requires Java version 7 or higher.

For compilation and execution

When using the Java interface, the Java wrapper jar\PRNA.jar needs to be on the CLASSPATH. You can do this by either adding it to the environment variable CLASSPATH, or by specifying it using the switch -classpath:

```
javac -classpath ".;C:\Program Files\PDF Tools AG\jar\PRNA.jar" ^
 sampleApplication.java
```

For execution

Additionally, the library PdfPrintAPI.dll needs be in one of the system's library directories[4] or added to the Java system property java.library.path. You can add the library by either adding it dynamically at program startup before using the API, or by specifying it using the switch -Djava.library.path when starting the Java VM. Choose the correct subdirectory (x64 or Win32 on Windows) depending on the platform of the Java VM[5].

```
java -classpath ".;C:\Program Files\PDF Tools AG\PRNA.jar" ^
 "-Djava.library.path=C:\Program Files\PDF Tools AG\bin\x64" sampleApplication
```

2.4.3 .NET interface

The 3-Heights® PDF Printer API does not provide a pure .NET solution. Instead, it consists of a native library and .NET assemblies, which call the native library. This has to be accounted for when installing and deploying the tool.

It is recommended that you use the NuGet package. This ensures the correct handling of both the .NET assemblies and the native library.

---

[3] Otherwise, you get the following message: LoadLibrary("PdfPrintAPI.dll") failed - The specified module could not be found.

[4] On Windows defined by the environment variable PATH, and on Linux defined by LD_LIBRARY_PATH.

[5] If the wrong data model is used, there is an error message similar to this: "Can't load IA 32-bit .dll on a AMD 64-bit platform"

Alternatively, the files in the ZIP archive can be used directly in a Visual Studio project targeting .NET Framework 2.0 or later. To achieve this, proceed as follows:

The .NET assemblies (`*NET.dll`) are added as references to the project; they are needed at compile time. `PdfPrintAPI.dll` is not a .NET assembly, but a native library. It is not added as a reference to the project. Instead, it is loaded during execution of the application.

For the operating system to find and successfully load the native library `PdfPrintAPI.dll`, it must match the executing application's bitness (32-bit versus 64-bit) and it must reside in either of the following directories:

- In the same directory as the application that uses the library
- In a subdirectory `win-x86` or `win-x64` for 32-bit or 64-bit applications, respectively
- In a directory that is listed in the PATH environment variable

In Visual Studio, when using the platforms "x86" or "x64", you can do this by adding the 32-bit or 64-bit `PdfPrintAPI.dll`, respectively, as an "existing item" to the project, and setting its property "Copy to output directory" to true. When using the "AnyCPU" platform, make sure, by some other means, that both the 32-bit and the 64-bit `PdfPrintAPI.dll` are copied to subdirectories `win-x86` and `win-x46` of the output directory, respectively.

2.4.4 C interface

- The header file `Printer_c.h` needs to be included in the C/C++ program.
- The library `PdfPrintAPI.lib` needs to be linked to the project.
- The dynamic link library `PdfPrintAPI.dll` needs to be in a path of executables (e.g. on the environment variable %PATH%).

2.5 Uninstall, Install a new version

If you have used the ZIP file for the installation, undo all the steps done during installation, e.g. de-register using `regsvr32.exe /u`, delete all files, etc.

Installing a new version does not require you to previously uninstall the old version. The files of the old version can directly be overwritten with the new version.

2.6 Note about the evaluation license

With the evaluation license, the 3-Heights® PDF Printer API automatically adds a watermark to the print-out.

2.7 Special directories

2.7.1 Directory for temporary files

This directory for temporary files is used for data specific to one instance of a program. The data is not shared between different invocations and is deleted after termination of the program.

The directory is determined as follows. The product checks for the existence of environment variables in the following order and uses the first path found:

Windows

1. The path specified by the %TMP% environment variable
2. The path specified by the %TEMP% environment variable

3. The path specified by the %USERPROFILE% environment variable
4. The Windows directory

2.7.2 Cache directory

The cache directory is used for data that is persistent and shared between different invocations of a program. The actual caches are created in subdirectories. The content of this directory can safely be deleted to clean all caches.

This directory should be writable by the application; otherwise, caches cannot be created or updated and performance degrades significantly.

Windows

- If the user has a profile:
  `%LOCAL_APPDATA%\PDF Tools AG\Caches`
- If the user has no profile:
  `<TempDirectory>\PDF Tools AG\Caches`

where `<TempDirectory>` refers to the Directory for temporary files.

2.7.3 Font directories

The location of the font directories depends on the operating system. Font directories are traversed recursively in the order as specified below.

If two fonts with the same name are found, the latter one takes precedence, i.e. user fonts always take precedence over system fonts.

Windows

1. `%SystemRoot%\Fonts`
2. User fonts listed in the registry key `\HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Fonts`. This includes user specific fonts from `C:\Users\<user>\AppData\Local\Microsoft\Windows\Fonts` and app specific fonts from `C:\Program Files\WindowsApps`
3. `Fonts` directory, which must be a direct subdirectory of where `PdfPrintAPI.dll` resides.

3 License management

The 3-Heights® PDF Printer API requires a valid license in order to run correctly. If no license key is set or the license is not valid, then most of the interface elements documented in Interface reference fail with an error code and error message indicating the reason.

More information about license management is available in the license key technote.

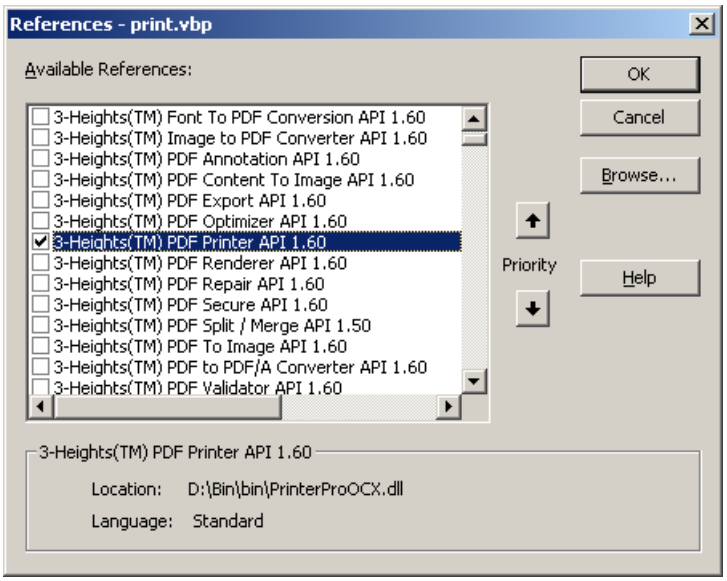
4 Programming interfaces

4.1 Visual Basic 6

After installing the 3-Heights® PDF Printer API and registering the COM interface (see Windows), you find a Visual Basic 6 example `print.vbp` in the directory `samples/VB/`. You can either use this sample as a base for an application, or you can start from scratch.

If you start from scratch, perform these steps:

1. Create a new Standard-Exe Visual Basic 6 project. Then include the 3-Heights® PDF Printer API component to your project.



2. Draw a new Command Button and optionally, rename it as appropriate.
3. Double-click the command button and insert the few lines of code below. All that you need to change is the path of the file name that is to be printed. This example is assuming you have installed a default printer.

Basic example:

```
Private Sub Command1_Click()
    Dim printer As New PDFPrinter
    printer.PrintFile"C:\pdf\input.pdf", ""
End Sub
```

And that's it - two lines of code. To modify the program to choose the document to be printed on a given printer and the options used, check out the code samples or see User guide.

4.2 ASP - VBScript

Here is a small sample of an ASP script using VBScript that prints the first page of a PDF document to the default printer, lists the name of the default printer, and the number of locally installed printers.

Example:

```
<%@ Language=VBScript %>
<%
 option explicit
 dim pdfPrint
 set pdfPrint = Server.CreateObject( "PrinterOCX.PDFPrinter")

 if not pdfPrint.PrintFile("c:\some.pdf", "", "", 1, 1) then
   Response.Write "<p>"
   Response.Write "Could not print file." & "<br>"
 end if

 Response.Write "<p>"
 Response.Write "Default printer: " & pdfPrint.GetDefaultPrinter & "<br>"
 Response.Write "Printer count: " & pdfPrint.GetPrinterCount & "<br>"
 Response.Write "</p>"
%>
```

4.3 .NET

There should be at least one .NET sample for MS Visual Studio available in the ZIP archive of the Windows version of the 3-Heights® PDF Printer API. The easiest to quickly start is to refer to this sample.

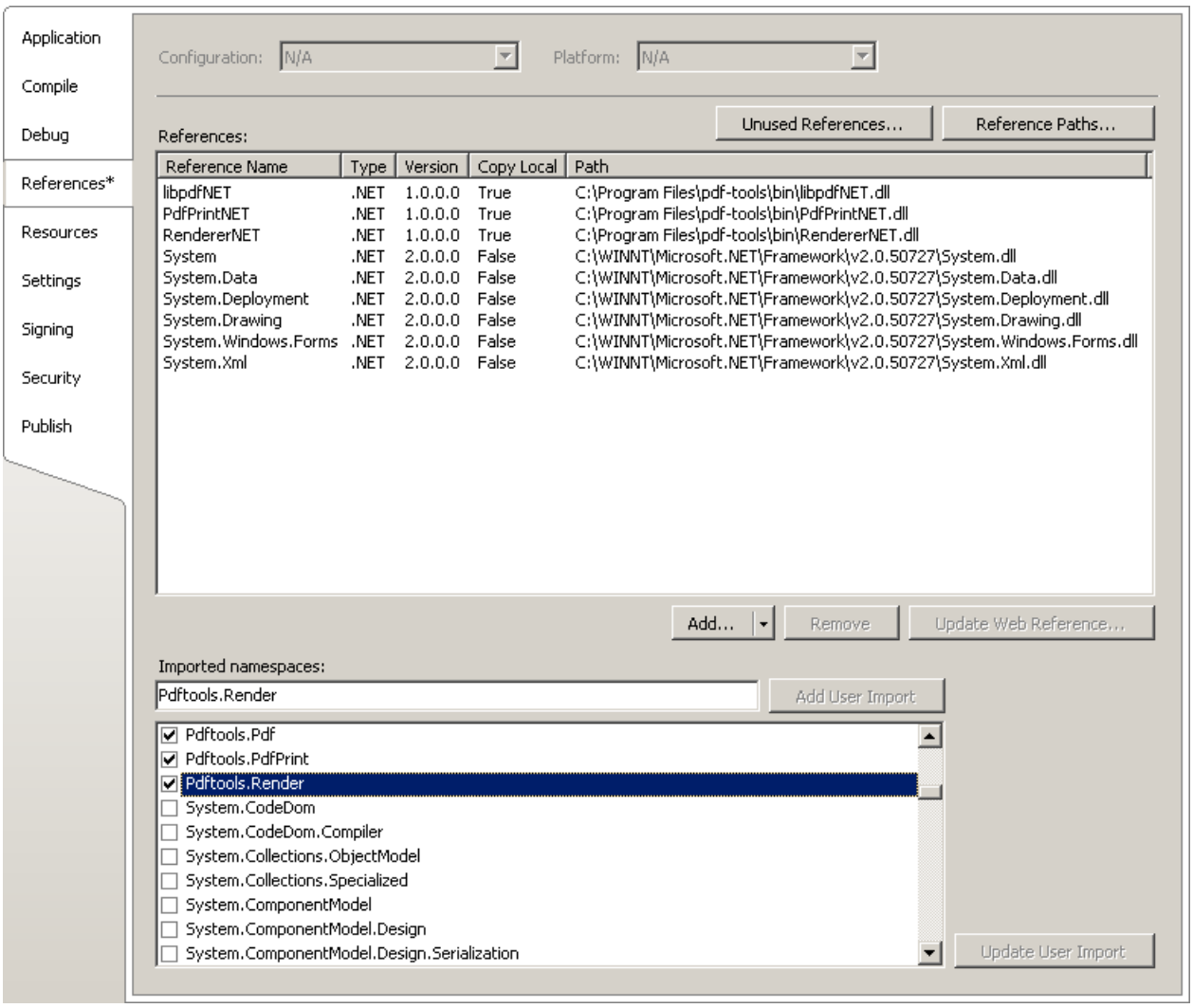
To create a new project from scratch, perform the following steps:

1. Start Visual Studio and create a new C# or VB project.
2. Add references to the NuGet package `PdfTools.PdfPrint 6.27.10`, as described in NuGet package.
3. Import namespaces (Note: This step is optional, but useful.)
4. Write your code.

Steps 3 and 4 are shown separately for C# and Visual Basic.

4.3.1 Visual Basic

3. Double-click “My Project” to view its properties. On the left hand side, select the menu “References”. The .NET assemblies you added before should show up in the upper window. In the lower window, import the namespaces `Pdftools.Pdf`, `Pdftools.PdfRenderer` , and `Pdftools.PdfPrint`.
   You should now have settings similar as in the screenshot below:



4. The .NET interface can now be used as shown below:

Example:

```
Dim printer As New Pdftools.PdfPrint.Printer
'Or if the namespace Pdftools.PdfPrint is
'imported:
' Dim printer As New Printer
If Not printer.OpenPrinter("HP LaserJet 4250 Series PCL 6") Then
  ...
If Not printer.Open("C:\\temp\\input.pdf") Then
  ...
printer.Orientation = PDFPrintOrientation.ePrint...
... OrientationPortrait
printer.BeginDocument("My Print Job")
printer.PrintPage(1)
...
```

4.3.2 C#

3. Add the following namespaces:

Example:

```
using Pdftools.Pdf;
using Pdftools.PdfRenderer;
using Pdftools.PdfPrint;
```

4. The .NET interface can now be used as shown below:

Example:

```
using (Printer printer = new Printer())
{
    if (!printer.OpenPrinter("HP LaserJet 4250 Series PCL 6"))
    { ... }
    if (!printer.Open("C:\temp\input.pdf"))
    { ... }
    printer.Orientation = PDFPrintOrientation.ePrintOrientationPortrait;
    printer.BeginDocument("My Print Job");
    printer.PrintPage(1);
    ...
}
```

4.3.3 Deployment

This is a guideline on how to distribute a .NET project that uses the 3-Heights® PDF Printer API:

1. The project must be compiled using Microsoft Visual Studio. See also .NET interface.
2. For deployment, all items in the project's output directory (e.g. `bin\Release`) must be copied to the target computer. This includes the 3-Heights® PDF Printer API's .NET assemblies (`*NET.dll`), as well as the native library (`PdfPrintAPI.dll`) in its 32 bit or 64 bit version or both. The native library can alternatively be copied to a directory listed in the PATH environment variable, e.g. `%SystemRoot%\System32`.
3. It is crucial that the native library `PdfPrintAPI.dll` is found at execution time, and that the native library's format (32 bit versus 64 bit) matches the operating system.
4. The output directory may contain multiple versions of the native library, e.g. for Windows 32 bit, Windows 64 bit, MacOS 64 bit, and Linux 64 bit. Only the versions that match the target computer's operating system need be deployed.
5. If required by the application, optional DLLs must be copied to the same folder. See Deployment for a list and description of optional DLLs.

4.3.4 Troubleshooting: TypeInitializationException

The most common issue when using the .NET interface is that the correct native DLL `PdfPrintAPI.dll` is not found at execution time. This normally manifests when the constructor is called for the first time and an exception of type `System.TypeInitializationException` is thrown.

This exception can have two possible causes, which you distinguish by the inner exception (property `InnerException`):

`System.DllNotFoundException` Unable to load DLL `PdfPrintAPI.dll`: The specified module could not be found.

`System.BadImageFormatException` An attempt was made to load a program with an incorrect format.

The following sections describe in more detail how to resolve these issues.

Troubleshooting: DllNotFoundException

This means that the native DLL `PdfPrintAPI.dll` could not be found at execution time.

Resolve this by performing one of these actions:

- Use the NuGet package.
- Add `PdfPrintAPI.dll` as an existing item to your project and set its property "Copy to output directory" to "Copy if newer", or
- Add the directory where `PdfPrintAPI.dll` resides to the environment variable `%Path%`, or
- Manually copy `PdfPrintAPI.dll` to the output directory of your project.

Troubleshooting: BadImageFormatException

The exception means that the native DLL `PdfPrintAPI.dll` has the incorrect "bitness" (i.e. platform 32 vs. 64 bit). There are two versions of `PdfPrintAPI.dll` available in the ZIP archive: one is 32-bit (directory `bin\Win32`) and the other 64-bit (directory `bin\x64`). It is crucial that the platform of the native DLL matches the platform of the application's process.

(Using the NuGet package normally ensures that the matching native DLL is loaded at execution time.)

The platform of the application's process is defined by the project's platform configuration for which there are three possibilities:

AnyCPU This means that the application runs as a 32-bit process on 32-bit Windows and as 64-bit process on 64-bit Windows. When using AnyCPU, then the correct native DLL must be used, depending on the Windows platform. You can perform this either when installing the application by installing the matching native DLL, or at application start-up by determining the application's platform and ensuring the matching native DLL is loaded. The latter can be achieved by placing both the 32 bit and the 64 bit native DLL in subdirectories `win-x86` and `win-x64` of the application's directory, respectively.

x86 This means that the application always runs as 32-bit process, regardless of the platform of the Windows installation. The 32-bit DLL runs on all systems.

x64 This means that the application always runs as 64-bit process. As a consequence, the application will not run on a 32-bit Windows system.

5 User guide

This chapter explains how most standard procedures of the 3-Heights® PDF Printer API work. Most samples are described in Visual Basic, but the functionality and calling sequence is similar for other languages such as Java, C, or C#.

5.1 Basics

5.1.1 Printing

The 3-Heights® PDF Printer API provides a series of functions to create print jobs. Some of these functions reflect the native Windows calls used for printing. Others are calls to link print jobs, or to simply print a page. Most of these functions come in pairs and are also to be used in pairs.

`Open`, `Close`	Open and close a PDF documents or raster image document. These functions can be called before, during, or after a print job.
`OpenPrinter`, `ClosePrinter`	Open and close an installed Windows printer, e.g. `HP LaserJet 4250 PS` or `Canon iR2200-3300 PCL6`. `ClosePrinter` must be called after the print job is completed using `EndDocument`.
`BeginDocument`, `EndDocument`	Define the beginning and end of a single print job. `BeginDocument` must always be called after a printer was opened.
`BeginGroup`, `EndGroup`	These calls are optional. They mark the beginning and end of a series of linked jobs, i.e. all print jobs started within the same group are printed consecutively, and without being interrupted by other print jobs (e.g. from other printing applications). Only documents that are printed on the same printer may be included within a single group.
`PrintPage`	This function prints a selected single page of the currently opened document to the current print job.

The above functions can be used in a very flexible way as shown later.

Same document to multiple printers

You can open a PDF document and print some of its pages to a printer and some of its pages (or the same pages) to different printers. A call sequence for a scenario where the 1st page of a document is printed to a different printer compared to the 2nd and 3rd page is given below:

```
Open("C:\mydocument.pdf")
   OpenPrinter("HP LaserJet 4250 PS")
      BeginDocument("my print job")     ' = Name that shows up in Spooler
         PrintPage(1)
      EndDocument
   ClosePrinter
   OpenPrinter("Canon iR2200-3300 PCL6")
      BeginDocument("my print job")
         PrintPage(2)
```

```
         PrintPage(3)
      EndDocument
   ClosePrinter
Close
```

Multiple documents to same printer

In a scenario where multiple documents are printed to the same printer and the documents should be printed in a single print job is given below:

```
OpenPrinter("HP LaserJet 4250 PS")
   BeginDocument("my print job ")
      Open("C:\mydocument.pdf")
         PrintPage(1)
      Close
      Open("C:\mydocument2.pdf")
         PrintPage(1)
      Close
   EndDocument
ClosePrinter
```

This sample only prints the first page of every document. To print all pages, replace `PrintPage(1)` by a for-loop. For example:

```
For n = 1 to PageCount
   PrintPage(n)
Next n
```

Multiple print jobs to same printer

If a very large number of pages is to be printed (e.g. more than 1000), it may make sense to break them down into individual print jobs to optimize resources (e.g. memory). If these jobs still must be printed consecutively, they can optionally be linked together using the `...Group` functions as in the example below:

```
OpenPrinter("HP LaserJet 4250 PS")
   BeginGroup    ' This is optional
      BeginDocument("my print job")
         Open("C:\mydocument.pdf")
            PrintPage(1)    ' Print more pages here
         Close
      EndDocument
      BeginDocument("my print job ")
         Open("C:\mydocument2.pdf")
            PrintPage(1)    ' Print more pages here
         Close
      EndDocument
   EndGroup    ' This is optional
ClosePrinter
```

As we have seen there are various ways how these calls can be combined. The only limitations are:

- Calls should be used in pairs. Every `Open`/`Begin` call must have its corresponding `Close`/`End` call. Even though a call to Open first closes an open document, you should use the pairs properly.

- `BeginDocument` must always be used after `OpenPrinter`, since a print job requires a printer to be specified.

One document to one printer

For the special case where exactly one document is to be printed to one printer, the order of the printing calls is not relevant. For this particular case, there is a special function: `PrintFile`.

These functions takes all parameters directly: Document name, printer name, and optionally, the password for an encrypted PDF document, first page, and last page of the page range that is to be printed. `PrintFile` cannot be combined with any other function mentioned in this chapter. It is a stand-alone function that internally calls `Open`, `OpenPrinter`, `BeginDocument`, `PrintPage`, etc.

The following call opens a document, opens a printer, starts a print job, prints page 1 and 2, ends the print job, closes the printer, and closes the document.

```
PrintFile ("C:\mydocument.pdf", "HP LaserJet 4250 PS","", 1, 2)
```

5.1.2 Settings

To set properties such as duplex/simplex mode, bin, paper size, or to open a printer, the procedure is identical. A printer can have multiple duplex modes, bins or paper sizes. A host can have multiple printers. The steps to set any of these properties or open a printer are:

1. Ask how many there are: `GetDuplexModeCount`, `GetBinCount`, `GetPaperCount`, `GetPrinterCount`
2. Select one and get its name: `GetDuplexMode`, `GetBin`, `GetPaper`, `GetPrinter`
3. Perform a task: `Duplex`=…, `DefaultSource`=…, `PaperSize`=…, `OpenPrinter`

Settings must always be configured before printing. The procedure to print and retrieve and set the configuration is described in subsequent chapters.

5.2 Print a document using PrintFile

The `PrintFile` method is used to print pages of one PDF document to one printer. A password and the first and last page of the document, defining the page range, can be provided optionally.

`PrintFile` cannot be combined with any other functions other than printing settings (such as duplex, paper size, etc.).

Example: Print all pages

This sample prints all pages of the PDF document `C:\some.pdf` to the local printer "`HP LaserJet 4050 Series PCL`".

```
Private Sub PrintFile1_Click()
    Dim printer As New PDFPrinter
    printer.PrintFile "C:\some.pdf", "HP LaserJet 4050 Series PCL"
End Sub
```

An empty string for the printer name indicates that the Windows default printer should be used.

An empty string as password (3rd parameter) indicates the document is not encrypted with a user password.

A page range can be defined using the optional parameters 4 and 5.

Example: Print a range of pages

This sample prints the first two pages of a file on the default printer:

```
Private Sub PrintFile2_Click()
    Dim printer As New PDFPrinter
    printer.PrintFile "C:\some.pdf", "", "", 1, 2
End Sub
```

To print multiple copies of a document, the number of copies is set before calling `PrintFile`. If the PDF document is encrypted with a user password, a password must be provided.

Example: Print an encrypted document

This sample print two copies of a user password-protected[6] encrypted PDF file to the default printer. Either the user or the owner password is `"mypassword"`.

```
Private Sub PrintFile3_Click()
    Dim printer As New PDFPrinter
    printer.Copies = 2
    printer.PrintFile "C:\some.pdf", "", "mypassword"
End Sub
```

5.3 Print documents using PrintPage

`PrintPage` requires a printer to be opened, a document to be opened, and a print job to be started. A possible call sequence using `PrintPage` is:

1. Open a printer
2. Start a print job
3. Open a PDF file
4. Print pages
5. Close the PDF file
6. (Optional) Open another file
7. (Optional) Print pages
8. (Optional) Close the PDF file
9. End the print job
10. Close the printer

Or using the according methods

```
OpenPrinter("HP Laser Jet 4250 Series PCL 6")
   BeginDocument("My Print Job")
      Open("C:\some.pdf")
         PrintPage(1)
      Close()
   EndDocument()
ClosePrinter()
```

The seven steps above can also be used in a different order. For example, a PDF file can be left open while the print job starts and ends, and a new print job for the same PDF is started (optionally, to a different printer). See also Basics.

[6] If a PDF document is encrypted with a user password, it means a password is required to open the document.

The seven steps are explained in the following samples.

5.4 List and open printers

The standard procedure to list all printers on a host is the following:

- Retrieve the number of printers on a selected host (n)
- List names of all printers (1 to n)

Example: List printers

This sample lists all printers on the local host.

```
Private Sub GetPrinter_Click()
   Dim printer As New PDFPrinter
   'an empty string stands for local host
   For i = 1 To printer.GetPrinterCount("")
      MsgBox "Printer " & i & ": " & printer.GetPrinter(i)
   Next i
End Sub
```

This sample writes the names of the printers to individual message boxes, which is useful for testing purposes. You can fill a combo box with the names retrieved this way, so the user can select one of the printers.

The standard procedure to select and open a printer on a host is the following:

- Retrieve the full name of a printer (see previous sample).
- Open the printer using its full name.

Example: Open printer

This sample opens the local printer `HP LaserJet 4050 Series PS`.

```
Private Sub OpenPrinter_Click()
   If Not printer.OpenPrinter("HP LaserJet 4050 Series PS") Then
      MsgBox "Error opening printer"
   End If
End Sub
```

The name of the default printer can be accessed using `GetDefaultPrinter`. A simpler way is to use an empty string as printer name, which stands for the Windows default printer. The sample below selects the Windows default printer and returns an error message when the printer is not available.

Example: Open default printer

The following sample opens the Windows default printer.

```
Private Sub OpenDefaultPrinter_Click()
   If Not printer.OpenPrinter("") Then
      MsgBox "Error opening default printer"
   End If
End Sub
```

5.5 Start a print job

A print job can be started using `BeginDocument`. The parameter of `BeginDocument` is the name of the print job. To define the end the print job, use `EndDocument`.

The method `PrintFile` starts a print job, prints, and ends the print job. It cannot be combined with other calls, such as `BeginDocument` or `EndDocument`.

5.5.1 Print job

A print job is a series of pages that are printed as one job. All pages of a print job are printed before the next print job starts.

The order in which individual print jobs are processed by the printer device is defined by the spooler. The order in which print jobs are created doesn't have to be the same order as they are printed. Small print jobs may receive higher priority and can overtake large print jobs. An exception to this are linked print jobs.

5.5.2 Linked print jobs

You can fix the order in which pages are printed by two ways:

1. Create a single print job that contains all pages. These pages can come from different documents.
2. Build a chain of individual print jobs and print them as linked print jobs.

The call sequence to link two print jobs together is as shown below:

1. Open printer
   A. Begin group
      a. Begin document
         i. Open file
         ii. Print pages
         iii. Close file
         iv. Optionally repeat i-iii
      b. End document
      c. Optionally repeat a-b
   B. End group
2. Close printer

5.5.3 Guidelines

You should keep the size of print jobs at a reasonable level. There are advantages and disadvantages for small and large print jobs.

- Small print jobs have the advantage of releasing resources quickly.
  For example, all fonts used in a print job are locked by the printer. If a document contains very large amounts of fonts or other resources, the system may run out of resources and fail to print the job after a certain amount of pages.
- Large print jobsd have the advantage of reducing overhead.
  For example, a font only needs to be sent once per print job. A few large print jobs are clearer arranged in a printer queue than hundreds of small jobs.

For documents with simple pages, you should bundle a maximum of around 1000 pages into a single print job. For more complex pages (complex vector drawings, large format newspapers, etc.), the bundle size should be reduced to around 100 pages.

5.6 Open a PDF document

Documents can be opened either from file using the Open method or from memory using the OpenMem method.

Example: Open PDF from file system

```
Private Sub Open_Click()
   Dim printer As New PDFPrinter
   printer.Open(App.Path & "\in.pdf")
End Sub
```

OpenMem is usually used when a PDF document is already available in memory, e.g. is read from a data base or is passed in-memory from another application.

The following example shows how a PDF document can be opened from memory by reading it from file and writing it in a byte array. The byte array is then read using OpenMem.

Example: Open PDF from memory

```
Private Sub OpenMem_Click()
   Dim printer As New PDFPrinter
   Dim bChar() As Byte
   Dim lFileLenght As Long
   Open App.Path & "\input.pdf" For Binary As #1
      lFileLenght = LOF(1)
      ReDim bChar(lFileLenght - 1)
      Get #1, , bChar
   Close #1
   printer.OpenMem bChar
End Sub
```

Using OpenMem is especially useful if the file is already in memory. This can either happen when the application just created the file or if the same file is used multiple times.

5.7 Print a specific page of a document

Print a specific page of a PDF document. The pre-conditions are that the PDF document was opened previously, there is a connection to a printer and a print job has been started.

Example: Print a specific page

```
Private Sub PrintPage_Click()
   Dim printer As New PDFPrinter
   'Open the default printer and the file
   '"\in.pdf"
   printer.OpenPrinter ""
   If Not printer.Open(App.Path & "\in.pdf") Then Exit Sub
   'Start print job, and set its name.
   printer.BeginDocument "The in.pdf Document"
   'Print Page 1, any other page could be added at this point
   printer.PrintPage 1
   printer.EndDocument
```

```
   printer.Close
   printer.ClosePrinter
End Sub
```

5.8 List and select the paper bin

The steps to set the printer bin are:

1. Get the total number of bins.
2. Select a number and get its value-name.
3. Set the bin using the value-name or a default value.

The name of the bins returned by the GetBin function is retrieved from the printer. This name does not always match the physical bins of the printer device.

The following example lists all bins of the default printer.

Example: List bins

```
Private Sub GetBin_Click()
   Dim printer As New PDFPrinter
   printer.OpenPrinter printer.GetDefaultPrinter
   For i = 0 To printer.GetBinCount(printer.GetDefaultPrinter)-1
      MsgBox "Bin " & i & ": " & printer.GetBin(i)
   Next
   printer.ClosePrinter
End Sub
```

The GetBin method returns a string. The first five characters of this string contain a number (possibly with leading blanks). This number is the number that is to be set when selecting the bin.

The numbers of the Windows default bins are listed in the Appendix. Either the Windows default bins or a custom bin can be used. The bin number is specified using the DefaultSource property.

Example: Set bin

Let's assume one of the strings returned by the GetBin_Click example above is " 259 Tray 1". The first five characters (blank, blank, 2, 5, 9) are to be converted to a number (259). This is the value to which the property DefaultSource needs to be set to.

```
Private Sub SetBin_Click()
   Dim printer As New PDFPrinter
   printer.DefaultSource = 259
End Sub
```

5.9 Duplex modes

The default values for Windows duplex modes are:

1	Simplex
2	Vertical duplex
3	Horizontal duplex

Example: Enable duplex

```
printer.Duplex = 2
```

On most printers, the three default values work fine. Experience has shown there are virtually no printers that support custom duplex modes only.

If you have one of these printers, use the GetDuplexModeCount method to receive the total number of custom duplex modes and then the GetDuplexMode function to actually receive the value-name. For sample code, look at the samples in List and select the paper bin.

The procedure to set the duplex mode is similar.

5.10 Set the paper size

The steps to set the paper size are:

- Get the total number of paper sizes
- Select a number and get its value-name
- Set the paper size using the name-value or a default value

The 118 Windows default paper sizes are listed in the Appendix.

For sample code, look at the samples in List and select the paper bin. The procedure to set the duplex mode is similar.

5.11 Get page dimensions

To receive the dimensions of a page of the PDF, the page must previously be chosen with the PageNo property. Then its dimensions can be accessed using the PageWidth and PageHeight properties.

Example: Retrieve Page Dimensions.

```
Private Sub PaperSize_Click()
   Dim printer As New PDFPrinter
   printer.Open ...
   For i = 1 To printer.PageCount
      printer.PageNo = i
      MsgBox "Page " & i & " width=" & printer.PageWidth _
        & " height=" & printer.PageHeight
   Next i
End Sub
```

5.12 Place a watermark

There is a series of watermark-related properties and methods that need to be set to place a watermark. The basic call sequence to add watermarks goes like this:

1. Set the desired watermark properties (WatermarkXPos, WatermarkYPos, WatermarkFontSize, etc.)
2. Add the watermark using either AddWatermarkText or AddWatermarkImage.
3. Repeat steps 1 and 2 in order to add multiple watermarks.
4. Print the page using PrintPage.
5. Delete watermarks using DeleteWatermarks in case the watermarks need to be deleted for the next page.

The above steps are shown in the Visual Basic 6 example below.

Example: Set multiple watermarks

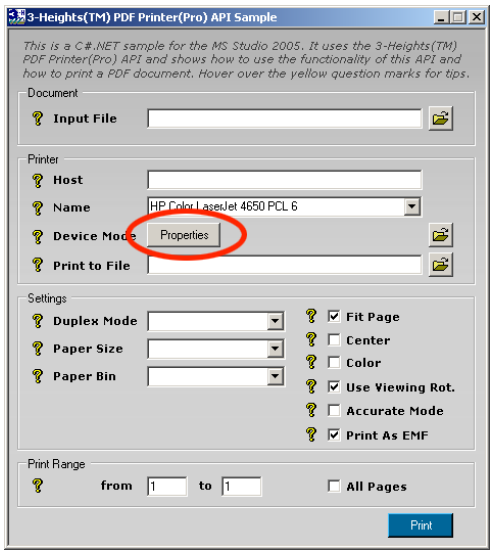
This sample shows how to set watermark on page 1 and how to remove it when printing page 2.

```
'1. Set the following properties in order to place a watermark.
'Set the position with 0/0 being in the upper left corner.(required)
    printer.WatermarkXPos = 100
    printer.WatermarkYPos = 200
'Set the rotation angle in radian. (optional)
    Dim angle As Single angle = 60 'degrees
    printer.WatermarkAngle = angle / 180 * 3.14156 'transform to radian
'Set the color in RGB. (optional)
    Dim red As Single, green As Single, blue As Single, color As Long
    red = 255
    green = 0
    blue = 0
    color = red + green * 256 + blue * 256 * 256 ' set color to red
    printer.WatermarkColor = color
' Set the font and font size. (optional)
    printer.WatermarkFontName = "Helvetica"
    printer.WatermarkFontSize = 80

'2. Add the watermark. (required)
    printer.AddWatermarkText "My Watermark"

'3. Repeat the above steps to add more watermarks. (optional)
    printer.WatermarkYPos = 300
    printer.AddWatermarkText "Another Watermark"

'4. Print a page. (required)
    printer.PrintPage 1

'5. Delete the watermark and print another page without watermark. (optional)
    printer.DeleteWatermarks
    printer.PrintPage 2
```

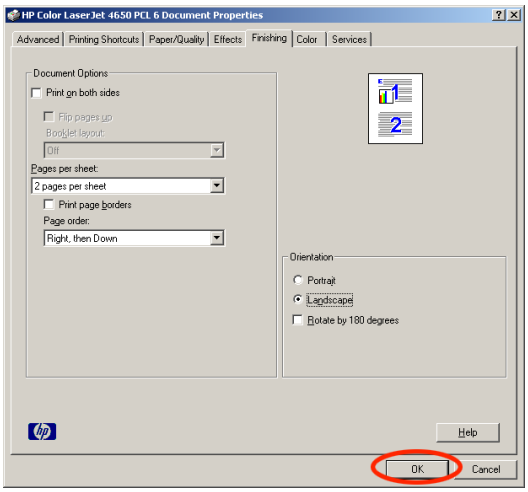
5.13 Using the device mode

Certain printers provide custom features such as stapling or printing multiple pages on one paper. These are a non-standard functionalities, i.e. not all printers support them. Therefore, they cannot be set using a property of the 3-Heights® PDF Printer API, but require a device mode.

1. A device mode is created using the `EditDevMode` function of the 3-Heights® PDF Printer API. This call opens the printer's GUI and allows the user to define any property the printer supports. In the C# sample, application is done by selecting the desired printer in the dropdown box "Name" and pressing the button "Properties".



2. This opens the printer's GUI as shown in the next screenshot.



3. In this dialog, apply the required settings (e.g. 2 pages per sheet in Landscape).
4. When done, press the button "OK". These settings are now binary, saved in the Printer API's property `DevMode`. This data is still volatile, i.e. after closing the application, it is lost. To save the data onto a file, see in the C# sample.
5. Certain features in the device mode require the network printing architecture of Windows. To ensure this, set `DataType` to "EMF".
6. You only need to create the device mode once. To automatically use the settings defined in the device mode, load it before printing (also described in the C# sample). If no device mode is loaded in the Printer API, the default device mode is used from the current printer's settings.

The created device mode is printer driver specific. A device mode can only be used with the printer model and driver that created it.

You can set a device mode in addition to setting standard printing properties of the 3-Heights® PDF Printer API, which overwrites the values in the device mode.

5.14 Using the options property

The `Options` property can be used to set various flags (see `Options`).

Options can be enabled (bitwise or) or disabled (bitwise and not). The default value is set to `eOptionBanding + eOptionTrueType + eOptionHighQuality + eOptionPrint`.

To enable or disable a particular flag, a code like the sample below can be used. This ensures that resetting a flag does not change the values of other flags.

Example: Visual Basic 6

```
'Enable Banding
printer.Options = printer.Options Or eOptionBanding
'Disable Banding
printer.Options = printer.Options And Not eOptionBanding
```

Example: C/C++

```
int iOptions = PDFPrnGetOptions(pDocument);
// Enable Banding
PDFPrnSetOptions(pDocument, iOptions | eOptionBanding);
// Disable Banding
PDFPrnSetOptions(pDocument, iOptions & ~eOptionBanding);
```

Example: C#

```
// Enable Banding
PdfViewer.Options |= (int) eOptionBanding;
// Disable Banding
PdfViewer.Options &= ~(int) eOptionBanding;
```

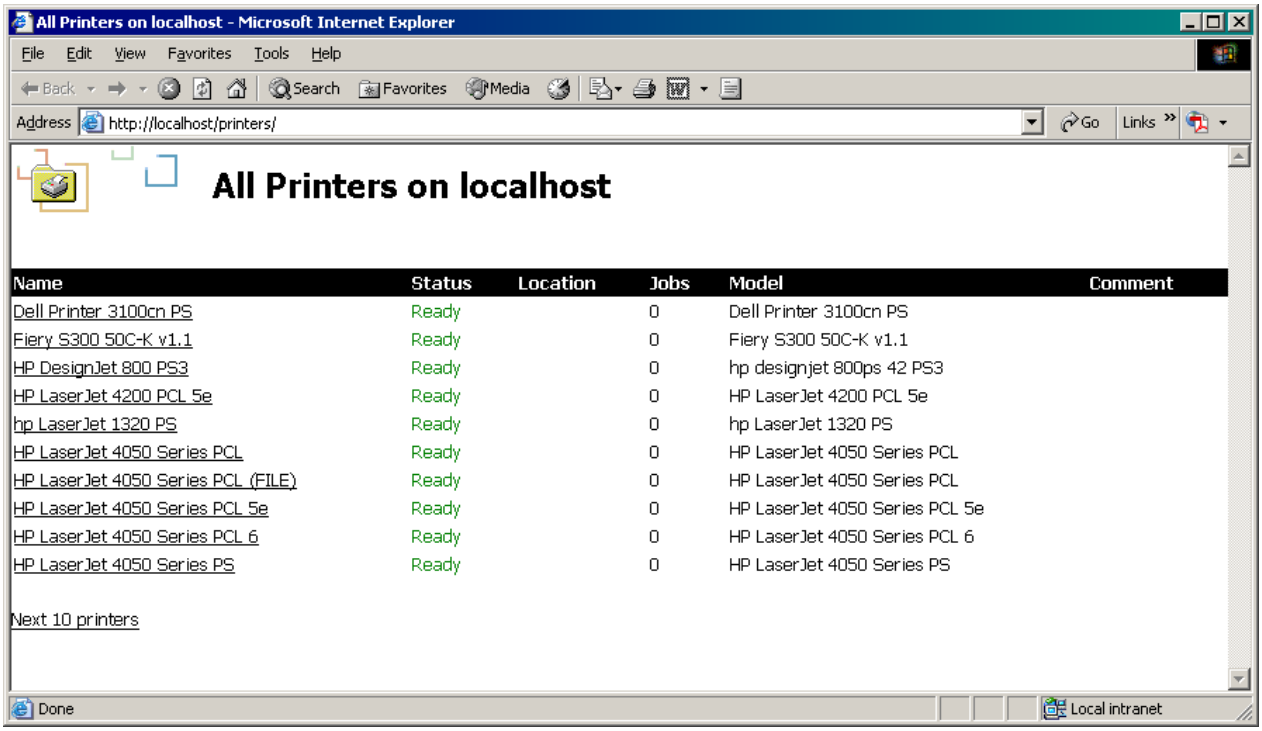
5.15 Internet printing

Printing via HTTP instead of the NetBIOS protocol requires the following three steps:

- Retrieve the name of the shared printer on the server.
- Provide the network location of the printer to the client.
- Select the URL as printer name.

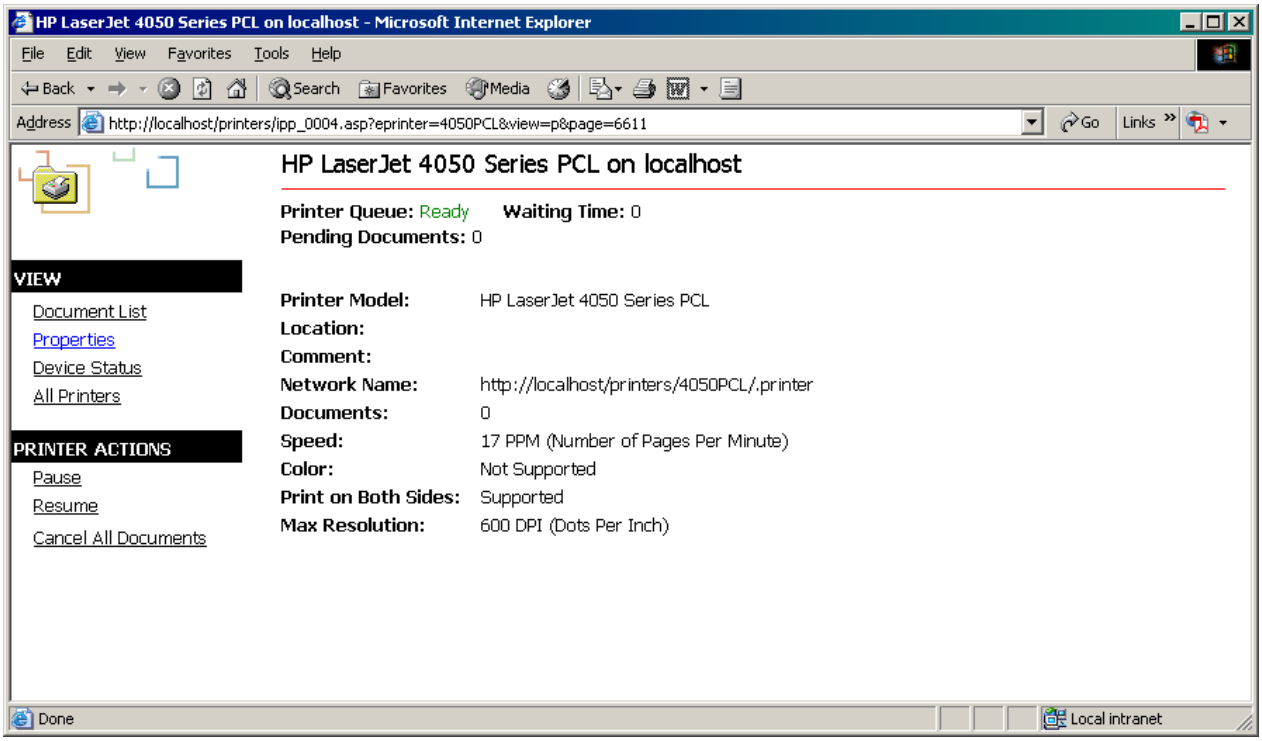
5.15.1 Retrieve the printer name

On the server where the printer is shared, open an Internet Explorer window and type `http://localhost/printers`. Instead of `localhost`, you also write the actual name of the server. This also works on the client if it is authorized to access the server.



This lists the available printers. Click on the one to which you want to print via HTTP.

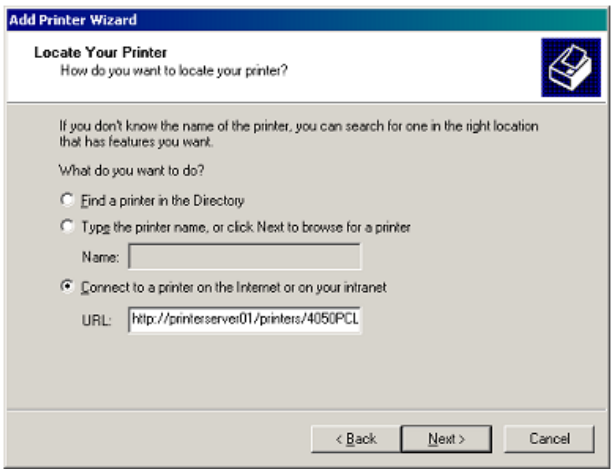
Then click “Properties” on the left hand side. You should see the properties, including the network name of the printer.



The URL then can be something like this: `http://localhost/printers/4050PCL/.printer`. `localhost` needs now to be replaced with the real name of the server, so the name could be: `http://printerserver01/printers/4050PCL/.printer`.

5.15.2 Set up the client

Start the “Add Printer” wizard on the client system. Select “Network Printer”, and then “Connect to a printer on the Internet or on your intranet”. As a URL, provide the “Network name” retrieved previously.



This step is required and ensures the client system can communicate with the printer on the server. It does not install a printer driver.

5.15.3 Connect to a printer via HTTP

When a printer is installed as described previously, it can be accessed via HTTP instead of NetBIOS. The corresponding command on the client looks like this:

```
-p "\\http://printerserver01\HP LaserJet 4050 Series PCL" input.pdf
```

(Note the two backslashes before the `http`.)

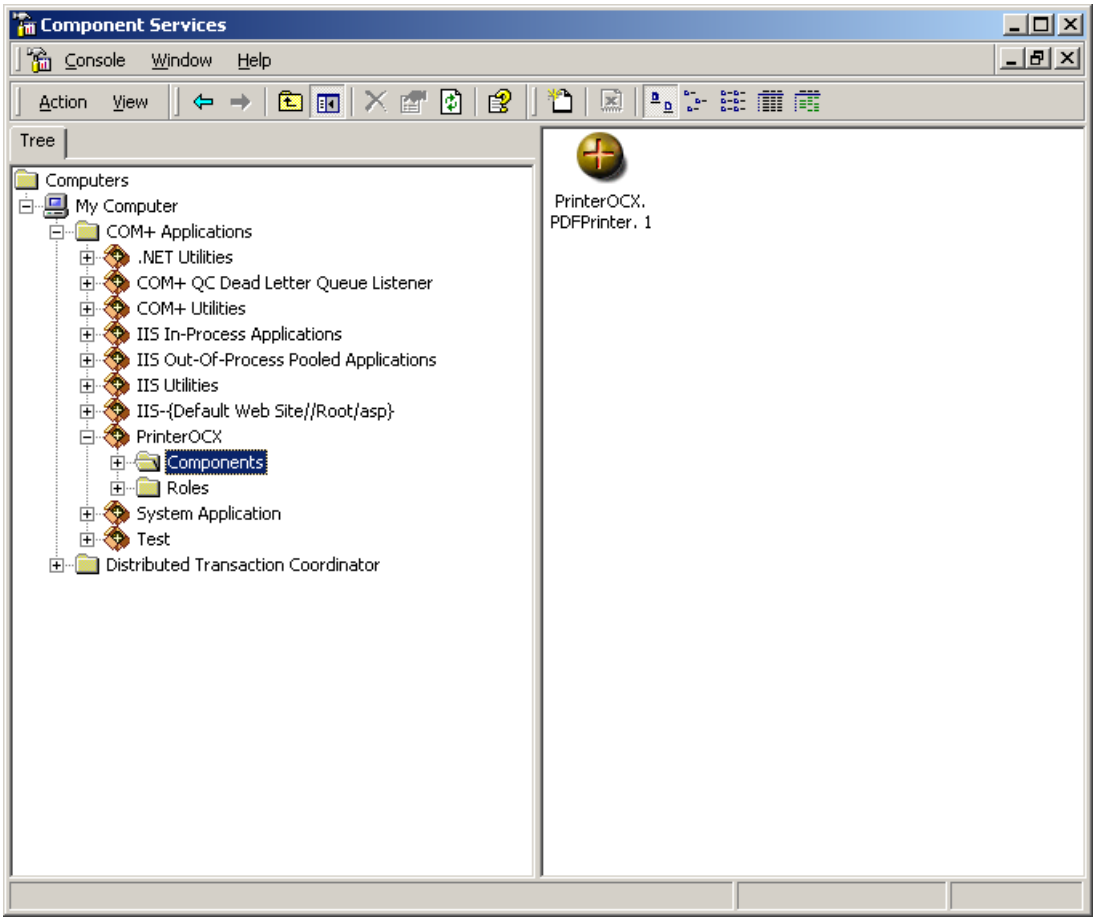
Keep in mind that using a printer via an Internet connection as described above may be unstable.

5.16 Creating a COM+ application

A COM+ application can resolve issues related to security when using the COM interface.

To create a new COM+ application for the 3-Heights® PDF Printer API, proceed as follows:

- Go to “Start” →“Settings” →“Control Panel” →“Administrative Tools” →“Component Services”.
- Right-click the COM+ Applications icon and create a new application.
- Select “Create an empty Application”.
- Name it, for example “PdfPrintAPI”, or “PrinterOCX”, select Server Application.
- Select the user under which the application should run. This account should have access to resources that needs be accessed such as documents or printers.
- By now, you should have something looking like this:



- Now right-click the icon Components and create a new component.
- Select “Install new Component”.
- Browse for the `PdfPrintAPI.dll`.
- Classes and interfaces should be found, click “Next”.

5.17 Color profiles

A PDF document may contain graphical objects using various different color spaces and the printout of 3-Heights® PDF Printer API may yet use another color space. Therefore often colors have to be converted between different color spaces.

For calibrated color spaces (such color spaces with an associated ICC color profile), the color conversion is well defined. For the conversion of uncalibrated device color spaces (DeviceGray, DeviceRGB, DeviceCMYK), however, the 3-Heights® PDF Printer API requires apropriate color profiles. Therefore, it is important that the profiles are available and that they describe the colors of the device your input documents are intended for.

> **Note:** When setting an alternative color management system such as Neugebauer, no color profiles are required.

If no color profiles are available, default profiles for both RGB and CMYK are generated on the fly by the 3-Heights® PDF Printer API.

5.17.1 Default color profiles

If no particular color profiles are set, default profiles are used. For device RGB colors, a color profile named "sRGB Color Space Profile.icm" and for device CMYK, a profile named "USWebCoatedSWOP.icc" are searched for in the following directories:

Windows

1. %SystemRoot%\System32\spool\drivers\color
2. directory Icc, which must be a direct subdirectory of where the PdfPrintAPI.dll resides.

5.17.2 Get other color profiles

Most systems have pre-installed color profiles available. For example, on Windows at %SystemRoot%\system32\spool\drivers\color\. Color profiles can also be downloaded from the links provided in the directory bin\Icc\ or from the following websites:

- https://www.pdf-tools.com/public/downloads/resources/colorprofiles.zip
- https://www.color.org/srgbprofiles.html

5.18 Fonts

PDF documents may contain both embedded and non-embedded fonts. When printing non-embedded fonts, the best result can be achieved, if the font is available on the system. Therefore, it is important to make sure the Font directories contain all fonts required.

For more information on how to deal with font issues, please refer to Font and text issues.

Note that on Windows, when a font is installed, it is by default installed only for a particular user. It is important to either install fonts for all users, or make sure the 3-Heights® PDF Printer API is run under that user and the user profile is loaded.

5.18.1 Font cache

A cache of all fonts in all Font directories is created. If fonts are added or removed from the font directories, the cache is updated automatically.

In order to achieve optimal performance, make sure that the cache directory is writable for the 3-Heights® PDF Printer API. Otherwise, the font cache cannot be updated and the font directories have to be scanned on each program startup.

The font cache is created in the subdirectory <CacheDirectory>/Installed Fonts of the Cache directory.

5.18.2 Font configuration file fonts.ini

The font configuration file is optional. It can be used to control the mapping of fonts used in the PDF to fonts pre-installed on the system.

The file fonts.ini must reside at the following location :

Windows: In a directory named Fonts, which must be a direct subdirectory of where PdfPrintAPI.dll resides.

It consists of two sections: [fonts] and [replace]. Both sections are used to map fonts in the PDF to fonts in the installed font collection on the operating system. This comes into play when the font in the PDF document does not have an embedded font program, or the embedded font is not usable.

The mapping only works if the font types of the specified fonts are matching; for example, if the font in the PDF is a symbolic font, such as “Symbol” or “ZapfDingbats”, the mapped font must be symbolic too.

The section `[fonts]` is only considered if the font-matcher does not find an appropriate font among the existing installed fonts. It is suggested to only use this section.

The section `[replace]` is stronger and applied before the font-matcher. This means a font will be replaced as defined, even if the correctly installed font is available on the system.

Syntax: The syntax of the mapping file is as follows

```
[fonts]
PDF_font_1=installed_font_1{,font_style}
PDF_font_2=installed_font_2{,font_style}
[replace]
PDF_font_n=installed_font_n{,font_style}
```

`PDF_font_*` is the name of the font in the PDF.

This name can be found in one of the following ways:

- Use any tool that can list fonts. Such as 3-Heights® PDF Extract or 3-Heights® PDF Optimizer. Ignore possible prefixes of font subsets. A subset prefix consists of 6 characters followed by the plus sign. For example “KHFOKE+MonotypeCorsiva”, in this case only use “MonotypeCorsiva” as font name in the mapping file.
- Open the document with Adobe Acrobat, use the “MarkUp Text Tool”, mark the text of which you would like to know the font name, right-click it, select “Properties...”

`installed_font_*` is the font family name of the installed font.

To retrieve this name, find the font in the Windows’ font directory and open it by double-clicking. The first line in the property window displays the font family name (this may vary depending on the operating system). The font family name does not include font styles; so an example of a font family name is “Arial”, but not “Arial Italic”.

`font_style` is an optional style that is added coma-separated after the font family name.

The style is always one word. Examples of font styles are “Italic”, “Bold”, “BoldItalic”. Omit the font style, if it is “Regular” or “Normal”.

Remove blanks from all font names, i.e. in both the `PDF_font_*` and the `installed_font_*`.

Example:

```
[fonts]
Ryumin-Light=MSMincho
GothicBBB-Medium=MSGothic
[replace]
ArialIta=Arial,BoldItalic
```

5.19 Error handling

Most methods of the 3-Heights® PDF Printer API can either succeed or fail depending on user input, the state of the PDF Printer API, or the state of the underlying system. It is important to detect and handle these errors to get accurate information about the nature and source of the issue at hand.

Methods communicate their level of success or failure using their return value. The return values to be interpreted as failures are documented in the Interface reference. To identify the error on a programmatic level, check the ErrorCode property. The ErrorMessage property provides a human readable error message, which describes the error.

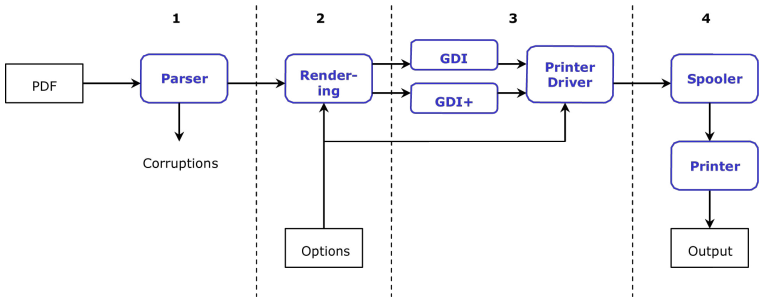
Example:

```
public Boolean Open(string file, string password)
{
  if (!printer.Open(file, password))
  {
    if (printer.ErrorCode == PDFErrorCode.PDF_E_PASSWORD)
    {
      password = InputBox.Show("Password incorrect. Enter correct password:");
      return Open(file, password);
    }
    else
    {
      MessageBox.Show(String.Format(
        "Error {0}: {1}", printer.ErrorCode, printer.ErrorMessage));
      return false;
    }
  }
  [...]
}
```

5.20 Printing workflow

5.20.1 Local

Below is a simplified flow chart of the 3-Heights® PDF Printer API workflow when printing locally (the printer is directly connected to workstation without printer server).



Parsing the input PDF document

A PDF document is passed to the 3-Heights® PDF Printer API. If the document exhibits minor corruptions, then these are fixed in this step. If there are major corruptions, then the file is rejected.

Rendering the pages

The PDF Printer API uses its own rendering engine. It runs independently from any third-party software. All version of PDF are supported. There are a few rare features of PDF that are not implemented at this time.

Create the spool file using GDI or GDI+ and the printer driver

The PDF Printer API has two rendering modes:

Fast rendering mode , which uses GDI (default)

Accurate rendering mode , which uses GDI+

Fast mode is optimized for creating small spool files. It supports direct PostScript data injection and an optimization for PCL creation.

Accurate mode applies image filters and is optimized for viewing on the monitor. It usually creates much larger spool files when used in combination with a printer due to its high resolution.

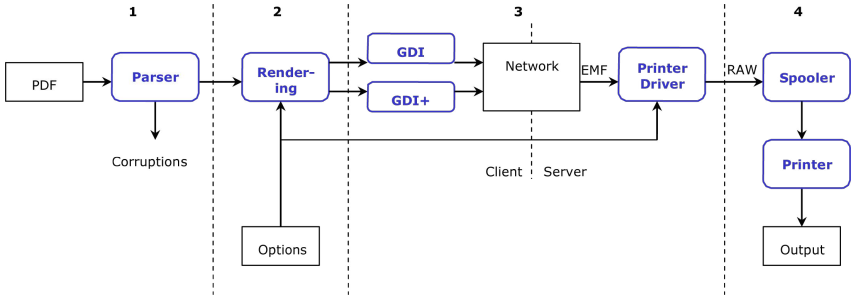
Spooler and printer device

Steps 1, 2, and 3 are under control of the 3-Heights® PDF Printer API or any other application that creates print jobs. Step 4 is performed by the spooler and the printer device.

If a print job is marked successful, it doesn't mean "The document could be printed", but rather "The print job is created, and is queued for printing by a printer device".

5.20.2 Network environment

When printing over a network, the flow chart slightly changes:



The printing application in step 2 always requires a printer driver.

A printer driver can be installed locally or remotely. If a printer driver is on a remote system, it is copied to the local system and temporarily installed for step 2.

6 Interface reference

Note: This manual describes the COM interface only. Other interfaces (C, Java, .NET) work similarly, i.e. they have calls with similar names and the call sequence to be used is the same as with COM.

6.1 `Printer` Interface

Properties have a default value. The default value can be read and can be overwritten. The value remains until a new value is set. Printer settings set via properties (unless set to -1) overrule the printer settings from the device mode.

6.1.1 `AbortDocument`

Method: `Boolean AbortDocument()`

Abort the current print job. The currently processed page is finished. Any further pages are aborted. After calling `AbortDocument`, no further calls to `PrintPage` are done.

Returns:

`True` Print job was aborted successfully.

`False` Otherwise.

6.1.2 `AddWatermarkImage`

Method: `Boolean AddWatermarkImage(String FileName)`

Add an image or PDF watermark to the print job. The properties of the watermark (scaling, position) are taken from the current settings of the corresponding properties (`WatermarkScale`, `WatermarkXPos`, `WatermarkYPos` etc).

Multiple watermarks can be added using multiple calls to `AddWatermarkImage`. The watermark must be added before the page or document is printed.

Parameter:

`FileName` [`String`] The file name and optionally, the file path, drive, or server string according to the operating system's file name specification rules of an image.

Returns:

True Watermark image successfully added.

False Otherwise.

6.1.3 AddWatermarkText

Method: `Boolean AddWatermarkText(String Text)`

Add a textual watermark to the print job. The properties of the watermark (font, size, position, etc) are taken from the current settings of the corresponding properties (`WatermarkScale`, `WatermarkXPos`, `WatermarkYPos`, etc).

Multiple watermarks can be added using multiple calls to `AddWatermarkText`.

The watermark(s) must be added before the page or document is printed.

The text can contain placeholders, which can be used to insert document specific text. The following placeholders are supported:

%ds	short date
%dl	long date
%t	document title
%a	document author
%s	document keywords
%k	document subject

For example, the following text contains the document's title, author, and date in the watermark text:

```
"watermark for the document %t written by %a, printed on %ds."
```

Parameter:

Text `[String]` The watermark text.

Returns:

True Watermark text successfully added.

False Otherwise.

6.1.4 Bandsize

Property (get, set): `Long Bandsize`
Default: `1048576 (2^20)`

Set or get the size of the chunks in bytes when banding is applied (i.e. when `Options` = `eOptionBanding`).

6.1.5 BeginDocument

Method: `Boolean BeginDocument(String DocumentName)`

Start a new printer job. All pages within one print job are printed successively, e.g. cannot be interrupted by another print job. The printer must be previously chosen with `OpenPrinter`. During or before the beginning of the print job, a PDF or image document can be opened from file or memory and closed.

This method can be repeated if its return value is `False` to recover from failures in a network printing environment (see Printing in a network environment).

The end of the print job is marked with `EndDocument`.

Parameter:

`DocumentName` `[String]` The name of the print job

Returns:

`True` Successfully connected to printer and started a print job.

`False` Otherwise.

6.1.6 BeginGroup

Method: `Boolean BeginGroup()`

Start a new chain of linked print jobs. All subsequent calls to `BeginDocument` create print jobs, which are linked to each other (i.e. they are printed in sequential order). The end of the chain is marked by a call to `EndGroup`. The print job is paused until `EndGroup` is called.

This method can only be used if:

1. Printing to a spooler, i.e. in the “Advanced Options” of the printer, the “Spool print documents” option must be set.
2. The `Output` property must not be set to a file name.
3. The `PJL` property must not be used.
4. The user that prints must have permission to administer print jobs.
5. The printer must not be a shared printer.

Returns:

`True` Successfully started a chain of linked print jobs.

`False` Otherwise.

6.1.7 Center

Property (get, set): `Boolean Center`
Default: `False`

Set or get the center mode. When set to `True`, the document is horizontally and vertically centered on the page. When set to `False`, the document is printed to the upper left corner of the page.

6.1.8 Close

Method: `Void Close()`

Close the currently opened document. If the document is already closed, the method does nothing.

6.1.9 ClosePrinter

Method: `Boolean ClosePrinter()`

Close the connection to the printer. It deletes temporarily installed font files of embedded fonts.

Returns:

`True` The connection could successfully be closed.

`False` The connection could not be closed.

6.1.10 Collate

Property (get, set): `Integer Collate`
Default: `-1`

Set or get the collate mode. It only has an impact if two or more copies of the document are printed using the method `PrintFile`. The following collate modes are supported:

Value	Description
-1	Use printer default.
0	Repeat page mode (Default): (1, 1, …2, 2, …3, 3, …) In this mode, every page is repeated as many times as copies are selected, then the next page, etc. In this mode, the print is "sorted" by page.
1	Repeat document mode: (1, 2, 3, …1, 2, 3, …) In this mode, all pages of the first copy are printed, then all pages of the second copy, etc. In this mode, the print is "sorted" by document.

6.1.11 Color

Property (get, set): Integer Color
Default: -1

Set or get the color mode. Supported values are:

Value	Description
-1	Use printer default.
DMCOLOR_MONOCHROME (0)	Monochrome
DMCOLOR_COLOR (1)	Color

6.1.12 Copies

Property (get, set): Integer Copies
Default: -1

Set or get the number of copies. This property should be used in combination with PrintFile . If the value of this property is set to -1, the number of copies defined in the printer is applied.

6.1.13 CopyMode

Property (get, set): Long CopyMode

This property sets or gets the copy mode. It only has an impact if two or more copies of the document are printed using the PrintFile method.

Value	Description
`0`	Disable copy mode: The API delegates the handling of multiple copies to the printer driver. Every page is only printed once by the API. As a result, the size of the spool file remains small, even if the number of copies is increased. Disabling the copy mode requires that the driver can handle printing multiple copies.
`1`	Enable copy mode: The API prints every page of every copy of the document. This mode works for all printer drivers. For multiple copies of a document, the spool file becomes larger.

6.1.14 DataType

Property (get, set): `String DataType`
Default: `""`

Set or get the data type of the spool file. There are two valid data types: `"raw"` and `"emf"`.

`"raw"`	`"raw"` is with respect to the printer language. For example, if `"raw"` is used for a PCL printer, a PCL file is created and if `"raw"` is used for a Postscript printer, a Postscript file is created. For local printers, the `DataType` should be set to `"raw"`.
`"emf"`	If `"emf"` is used, an EMF[7] file is generated. The EMF file can be sent over a network and at its destination the (remote-) printer driver converts it to a raw file. For network printers, `"emf"` should be used and it comes with the following two advantages: ■ It uses less bandwidth to send the spool file over the network, because an EMF file is smaller than raw spool file. ■ The workload is balanced: on the host where the Printer API resides, the EMF file is generated; on the host where the printer resides, the EMF file is converted to a raw file.
`""`	If `DataType` is set to an empty string or `Nothing`, then the data type is inherited from the printer's setting of the current user. In any situation where the current user settings are not well defined (e.g. IIS), the `DataType` should be set explicitly to either `"raw"` or `"emf"`. Some printer drivers only allow the setting of additional options if the datatype is set to EMF or RAW explicitly. For these drivers, use `Nothing`.

6.1.15 DC

Property (get): `Long DC`

This property returns a handle to the device context. It is valid after a successful call to `BeginDocument` and until calling `EndDocument`.

6.1.16 DefaultSource

Property (get, set): `Integer DefaultSource`
Default: `-1`

The default source defines from which input tray the paper is selected. For default values, see Paper bins. There is no property to set the output paper tray. To set the output paper tray, use the device mode functionality.

Some printers ignore the value of this property. If `DefaultSource` has no effect on the paper bin chosen by the printer, sometimes the bin can be selected using the `MediaType` property.

6.1.17 DeleteWatermarks

Method: `Void DeleteWatermarks()`

Delete all current watermarks, which includes textual and image watermarks.

6.1.18 DevMode

Property (get): `Long DevMode`

Set or get the device mode of the currently open printer. The `EditDevMode` method modifies this property.

This property can be got or set only if a printer is open, i.e. in between `OpenPrinter` and `ClosePrinter` calls. See Using the device mode for more information on device modes.

6.1.19 Duplex

Property (get, set): `Integer Duplex`
Default: `-1`

Set or get the duplex mode. For Windows default values, see Duplex modes. It is suggested to use the default values 1, 2, and 3.

`-1`	Use printer default
`DMDUP_SIMPLEX (1)`	Simplex
`DMDUP_VERTICAL(2)`	Vertical duplex
`DMDUP_HORIZONTAL(3)`	Horizontal duplex

6.1.20 EditDevMode

Method: `Boolean EditDevMode(Long hwndParent)`

Open the printer properties dialog and allows the interactive modification of its settings (e.g. paper size, duplex, number of copies, etc.). Upon pressing the OK button in the dialog, the settings (device mode) are saved. Otherwise, they are discarded. A printer must be selected prior to editing the device mode.

Parameter:

`hwndParent` [Long] Handle to the parent window.

Returns:

`True` The device mode was successfully edited and the user pressed "OK". The edited device mode is now available in the DevMode property.

`False` Otherwise (e.g. user pressed "Cancel").

6.1.21 EmptyPage

Method: `Boolean EmptyPage(Double Width, Double Height)`

Insert an empty page of the given dimensions into the spool output. There must be an open printer connection, but there is no need for an open document before calling this method. Active watermarks are printed on the empty page. The page dimensions are used to scale and rotate the page and select a suitable bin if appropriate.

Parameters:

`Width` [Double] The width of the page in points

`Height` [Double] The height of the page in points

Returns:

`True` The page was successfully printed.

`False` Otherwise.

6.1.22 EndDocument

Method: `Boolean EndDocument()`

Define the end of the printer job. After calling `EndDocument`, the print job is no longer under the control of the 3-Heights® PDF Printer API.

When printing directly to a printer (i.e. not using a spooler), `EndDocument` means the entire print job is on the printer.

When using the spooler, `EndDocument` means the entire print job is in the queue of the spooler. It does not imply that it is already being printed.

Returns:

`True` The print job was submitted and the connection to the printer could successfully be closed.

`False` Otherwise.

6.1.23 EndGroup

Method: `Boolean EndGroup()`

Define the end of a chain of linked print jobs.

Returns:

`True` The end of the print job chain was set successfully.

`False` Otherwise.

6.1.24 ErrorCode

Property (get): `TPDFErrorCode ErrorCode`

This property can be accessed to receive the latest error code. This value should only be read if a function call on the PDF Printer API has returned a value, which signales a failure of the function (see Error handling). See also enumeration `TPDFErrorCode`. Pdftools error codes are listed in the header file `bseerror.h`. Please note that only few of them are relevant for the 3-Heights® PDF Printer API.

6.1.25 ErrorMessage

Property (get): `String ErrorMessage`

Return the error message text associated with the last error (see property `ErrorCode`). This message can be used to inform the user about the error that has occurred. This value should only be read if a function call on the PDF Printer API has returned a value, which signals a failure of the function (see Error handling)

> **Note:** Reading this property if no error has occurred can yield `Nothing` if no message is available.

.

6.1.26 Escape

Method: `Boolean Escape(Variant varData)`

This method is used to pass a binary (or text) string to the printer driver. It is cached and inserted in the printer data stream immediately after the GDI StartPage call without any interpretation as its content depends on the specific printer language.

Parameter:

`varData` [Variant] A binary or text string.

Returns:

`True` If the method completed successfully.

`False` Otherwise.

6.1.27 FitPage

Property (get, set): `Boolean FitPage`
Default: `False`

The fit page property defines how the PDF page should fit the paper size. Allowed values are:

`True`	The page is resized so that both page width and height fit on the printable part of the paper supported by the printer device. The ratio width to height remains unchanged.
`False`	The size of the page remains unchanged. If part of the content is outside the printable area (i.e. close to the border of the page) it is not be printed.

6.1.28 GetBin

Method: `String GetBin(Integer iBin)`

Return the value name of the input paper bin with number `iBin`. The returned string contains a number (1-5 digits) and a description of the bin. The number is the value that needs to be used to set the bin by the property `DefaultSource`.

Some printers store information about bins in the device mode. Therefore, it is recommended that you use `OpenPrinter` and optionally, set the `DevMode` before listing bins.

Parameter:

`iBin` [`Integer`] The bin number. When iterating over all bins, `iBin` runs from `0` to `GetBinCount()-1`.

> **Compatibility note:** In the COM interface of versions prior to 1.91.0.20, `iBin` runs from `1` to `GetBinCount`.

Returns:

`Value-Name` The value name for the bin with number `iBin`.

6.1.29 GetBinCount

Method: `Integer GetBinCount(String PrinterName)`

Return the total number of input paper bins on a printer. This method should be used prior to `GetBin`.

Parameter:

`PrinterName` [`String`] The name of the printer.

Returns:

The number of total bins for the printer.

6.1.30 GetDefaultPrinter

Method: `String GetDefaultPrinter()`

Return the name of the default printer, if there is a default printer installed on the system. If there is no default printer defined, it returns an empty string.

Returns:

The name of the default printer.

6.1.31 GetDuplexMode

Method: `String GetDuplexMode(Integer iDuplex)`

Return the value name of the duplex mode with number `iDuplex`.

Parameter:

`iDuplex` [Integer] The duplex mode number.

Returns:

`Value-Name` The value name for the duplex mode with number `iDuplex`.

6.1.32 GetDuplexModeCount

Method: `Integer GetDuplexModeCount(String PrinterName)`

Return the total number of duplex modes on a printer. This method should be used prior to `GetDuplexMode`.

Parameter:

`PrinterName` [String] The name of the printer.

Returns:

The number of total duplex modes for the printer.

6.1.33 GetErrorText

[Deprecated] Method: `String GetErrorText(TPDFErrorCode iErrorCode)`

Deprecated. Use `ErrorMessage` instead.

6.1.34 GetMediaType2

Method: `Integer GetMediaType2(Integer iMediaType)`

Get the media type value that can be used to specify the respective media type for printing using the `MediaType` property. Standard media type numbers are listed in the MSDN documentation.

Some printers store information about media types in the device mode. Therefore, it is recommended that you use `OpenPrinter` and optionally, set the `DevMode` before listing media types.

Parameter:

`iMediaType` [Integer] The media type index. When iterating over all media types, `iMediaType` runs from `0` to `GetMediaTypeCount()-1`.

Returns:

The media type value of the media type with number `iMediaType`.

6.1.35 GetMediaTypeCount

Method: `Void GetMediaTypeCount(String PrinterName)`

Returns the total number of supported MediaTypes of a printer. It should be used prior to `GetMediaType2` and `GetMediaTypeName`.

Parameter:

`PrinterName` `[String]` The name of the printer.

6.1.36 GetMediaTypeName

Method: `String GetMediaTypeName(Integer iMediaType)`

Returns the media type value and name of the media type with number `iMediaType`. See `GetMediaType2` and `GetMediaTypeCount`.

Parameter:

`iMediaType` `[Integer]` The media type index. When iterating over all media types, iMediaType runs from `0` to `GetMediaTypeCount()-1`.

6.1.37 GetPageDimensions

[Deprecated] Method: `GetPageDimensions(Float* Width, Float* Height)`

Deprecated, use `PageWidth` and `PageHeight` instead.

6.1.38 PageWidth

Property (get): `Float PageWidth`

The Size of a page. The page needs first to be selected with `PageNo`.

The dimensions for a PDF document are in PDF points. (72 points = 1 inch, 1 inch = 25.4 mm). This method can also be applied to raster images, such as TIFF, JPEG, PNG, etc. In this case, the resolution of the image is applied if existing. If the image does not provide a resolution, it is set to 96 DPI (dots per inch).

Examples:

1. A TIFF image 800 by 600 pixels has a resolution of 72 DPI: `PageWidth` returns 800 points.
2. A TIFF image 800 by 600 pixels has a resolution of 300 DPI: `PageWidth` returns 192 points.
3. For a JPEG image 800 by 600 pixels without an associated resolution a resolution of 96 DPI is assumed: `PageWidth` returns 600 points.

6.1.39 PageHeight

Property (get): `Float PageHeight`

The size of a page. The page needs first to be selected with `PageNo`.

The dimensions for a PDF document are in PDF points. (72 points = 1 inch, 1 inch = 25.4 mm). This method can also be applied to raster images, such as TIFF, JPEG, PNG, etc. In this case, the resolution of the image is applied if existing. If the image does not provide a resolution, it is set to 96 DPI (dots per inch).

Examples:

1. A TIFF image 800 by 600 pixels has a resolution of 72 DPI: `PageHeight` returns 600 points.
2. A TIFF image 800 by 600 pixels has a resolution of 300 DPI: `PageHeight` returns 144 points.
3. For a JPEG image 800 by 600 pixels without an associated resolution a resolution of 96 DPI is assumed: `PageHeight` returns 450 points.

6.1.40 GetPaper

Method: `String GetPaper(Integer iPaper)`

Return the value name of the paper size with number `iPaper`.

Some printers store information about papers in the device mode. Therefore, it is recommended that you use `OpenPrinter` and optionally, set the `DevMode` before listing papers.

Parameter:

`iPaper` `[Integer]` The paper number. When iterating over all papers, `iPaper` runs from `0` to `GetPaperCount()-1`.

> **Compatibility note:** In the COM interface of versions prior to 1.91.0.20, `iPaper` runs from `1` to `GetPaperCount`.

Returns:

`Value-Name` The value name for the paper with number `iPaper`.

6.1.41 GetPaperCount

Method: `Integer GetPaperCount(String PrinterName)`

Return the total number of paper sizes on a printer. This method should be used prior to GetPaper.

Parameter:

`PrinterName` [String] The name of the printer

Returns:

The total number of paper sizes on the printer.

6.1.42 GetPrinter

Method: `String GetPrinter(Integer iPrinter)`

Return the name of the printer with number `iPrinter`. To select a remote printer, use GetPrinterCount to specify the host prior to `GetPrinter`. `GetPrinter` then selects the printer with number `iPrinter`.

Parameter:

`iPrinter` [Integer] The number of the printer

Returns:

The name of the printer with number `iPrinter`.

6.1.43 GetPrinterCount

Method: `Integer GetPrinterCount(String Host)`

Return the total number of printers on a host. This method should be used prior to GetPrinter.

Parameter:

`Host` [String] The name of the host. For localhost, use an empty string (default).

Returns:

The total number of printers available on the host.

6.1.44 HANDLE

Property (get): `Long HANDLE`

Get the current printer handle. This property is valid after a successful call to `OpenPrinter` and until calling `ClosePrinter`. The handle, which is defined by the `OpenPrinter` call, is opened with the `PRINTER_ALL_ACCESS` access rights.

6.1.45 JobId

Property (get): `Long JobId`

Returns the ID of the current print job after `BeginDocument` has been called. This allows for managing the print job using Windows API functions.

6.1.46 LicenseIsValid

Property (get): `Boolean LicenseIsValid`
Static

Check if the license is valid.

6.1.47 MaxDPI

Property (get, set): `Integer MaxDPI`
Default: `INT_MAX`

Set the maximum DPI that is used for prerendered content sent to the printer. If not set, then the DPI configured in the printer driver is used.

The maximum DPI is only applied if either the `eOptionBitmap` option is used or if a page contains enough transparent content that the PDF Printer API decides to prerender the entire page. In all other cases, the device resolution advertised by the printer driver is used.

6.1.48 MaxPaper

Property (get, set): `Integer MaxPaper`
Default: `0`

Set the maximum paper size that is supported by the automatic paper size feature (`PaperSize` set to `-2`). Any paper size that exceeds the paper width or height is excluded. The paper sizes are represented by an inter value as returned in the first five characters by the `GetPaper` function; see also Paper bins.

Example:

If MaxPaper is set to 66 (A2), then larger paper sizes such as A1 are ignored by the automatic paper selection.

6.1.49 MediaType

Property (get, set): `Integer MediaType`
Default: `-1`

Set or get the MediaType. Supported values are:

Value	Description
`-1`	Use printer default
`DMMEDIA_STANDARD` (1)	Standard paper
`DMMEDIA_TRANSPARENCY` (2)	Transparency
`DMMEDIA_GLOSSY` (3)	Glossy paper

Media types supported by a printer can be enumerated using the `GetMediaTypeCount`, `GetMediaType2`, and `GetMediaTypeName` methods.

6.1.50 MinLineWidth

Property (get, set): `String MinLineWidth`
Default: `0`

Get or set the minimum line width in PDF points. In cases where lines are printed too thin, a minimum line width in PDF points can be defined. Any line is then printed with at least the defined minimum line width. As a result, thin and very thin lines can no longer be distinguished.

This option only affects lines. It has no influence when lines are drawn in a way other than using the PDF operators m and l (move to, line to). It does not affect text unless text is drawn with lines instead of using a font.

6.1.51 OffsetX, OffsetY

Property (get, set): `Long OffsetX`
Default: `0`

Property (get, set): `Long OffsetY`
Default: `0`

Set or get the X and Y-offset of the page on the paper. Units: 1/100 millimeters.

6.1.52 OMR

Property (get, set): `String OMR`
Default: `""`

The string specifies OMR markers that are printed on each succeeding page until the marker is changed or deleted. The syntax of the string is as follows:

Example: `"0, 20, 10, 4, 15, 0, 01110011"`

0 horizontal position of the first markers

20 vertical position of the first marker

10 horizontal extension of the marker

4 vertical extension of the marker

0 Markers are drawn from either top of page to bottom (1) or from bottom to top (0).

01110011 Array of Boolean numbers indicating whether the marker shall be present or not.

This property was introduced with version 1.91.6.0.

6.1.53 Open

Method: `Boolean Open(String Filename, String Password)`

Open a PDF file or raster image file, i.e. make the objects contained in the document accessible. If another document is already open, it is closed first.

Parameters:

Filename [String] The file name and optionally, the file path, drive or server string according to the operating systems file name specification rules.

Password [String] (optional) The user or the owner password of the encrypted PDF document. If this parameter is left out, an empty string is used as a default.

Returns:

True The file could be successfully opened.

False The file does not exist, it is corrupt, or the password is not valid. Use the ErrorCode and ErrorMessage properties for additional information.

6.1.54 OpenMem

Method: `Boolean OpenMem(Variant MemBlock, String Password)`

Open a PDF file or raster image file, i.e. make the objects contained in the document accessible. If a document is already open, it is closed first.

Parameters:

`MemBlock` [`Variant`] The memory block containing the PDF file given as a one-dimensional byte array.

`Password` [`String`] (optional) The user or the owner password of the encrypted PDF document. If this parameter is left out, an empty string is used as a default.

Returns:

`True` The document could be successfully opened.

`False` The document could not be opened, it is corrupt, or the password is not valid.

6.1.55 OpenPrinter

Method: `Boolean OpenPrinter(String PrinterName)`

Open a printer.

This method can be repeated if its return value is `False` to recover from failures in a network printing environment (see Printing in a network environment).

Parameter:

`PrinterName` [`String`] The name of the printer. The name is the same as shown on the Settings/Printer window. For example, `"HP LaserJet 4050 Series PS"`. It is not the same as the network name. Network printers could look like this: `"\\PrinterServer\HP LaserJet 4250 PCL 6"`.

Returns:

`True` The printer could be successfully opened.

`False` Otherwise.

6.1.56 Options

Property (get, set): `TPDFRendererOption Options`
Default: `eOptionBanding + eOptionTrueType + eOptionHighQuality + eOptionPrint`

Set or get a specific rendering option.

Use bitwise “OR” to add an option.

Use bitwise “AND NOT” to remove an option.

For more information on the options available in the 3-Heights® PDF Printer API and how to use the this property, see TPDFRendererOption.

6.1.57 Orientation

Property (get, set): Integer Orientation
Default: -1

Set or get the orientation of the paper. Allowed values are:

-2	Automatic
-1	Use printer setting (default)
DMORIENT_PORTRAIT (1)	Force portrait
DMORIENT_LANDSCAPE (2)	Force landscape

When not specified, the PDF Printer API uses the setting of the printer.

Compatibility note: In older versions, the default was set to automatic, which places the page on the paper such that it fits best.

6.1.58 Output

Property (get, set): String Output
Default: ""

Set or get the path to an output file. The output is then redirected to the specified file name. The selected port of the printer is ignored. If the string is empty, the output is not redirected to a file.

This property must be set before BeginDocument to redirect the print job.

This property must be set to an empty string if using BeginGroup and EndGroup.

6.1.59 Page

[Deprecated] Property (get, set): Long Page

Deprecated, use PageNo instead.

6.1.60 PageCount

Property (get): Long PageCount

Get the number of pages of an open document. If the document is closed or if the document is a collection (also known as PDF portfolio), then this property is `0`.

6.1.61 PageNo

Property (get, set): `Long PageNo`

Set or get the current page number in the PDF. Set the page number before retrieving information from this page, such as `PageWidth` and `PageHeight`.

The numbers are counted starting from `1` for the first page to the value of `PageCount` for the last page.

6.1.62 PaperSize

Property (get, set): `Integer PaperSize`
Default: `-1`

Get or set the paper size. The 118 Windows default paper sizes are listed in Paper sizes.

If the value is set to `-1`, the default paper configured in the printer is used.

If the value is set to `-2`, the 3-Heights® PDF Printer API calculates the paper sizes of the pages. It transmits the paper size to the printer, which then selects the paper from the appropriate bin automatically. Use the method `SetPaperList` in order to limit the allowed set of paper sizes used.

If the value is set to `-2`, the setting of the property `DefaultSource` is ignored.

Background information when using `-2`: There are printer drivers that return a generic result when asked which paper sizes they support (see `GetPaperCount` and `GetPaper`). That includes paper sizes that are not supported by the connected physical device, and are intended for another type of device, which shares the same driver. This may result in selecting a paper size that is too large and therefore not supported. As an undesired consequence of the too large paper size not being available, the printer selects the default paper size. To "manually" limit the paper sizes to those that are actually available, use the property `MaxPaper`.

6.1.63 PJL

Property (get, set): `String PJL`
Default: `""`

Allow PJL commands to be inserted directly into the header of the spool file and override identical PJL commands that are created by the printer. The PJL command can be passed in two ways:

1. As text file.
   In this case, the value of the property is set to the file path. The text file should be ANSI and contain one command per line. The last line should be empty.

   Example: for text file `C:\path\pjl.txt`

```
@PJL SET ECONOMODE=ON
```

```
@PJL SET RESOLUTION=300
```

Example to set PJL property (C#):

```
PJL = @"C:\path\pjl.txt";
```

2. As direct PJL command.

Example to set PJL property (C#):

```
PJL = "@PJL SET ECONOMODE=ON\n@PJL SET RESOLUTION=300\n";
```

To avoid setting any PJL commands, set this property to an empty string (default).

This property was introduced with version 1.91.6.0.

6.1.64 PrinterStatus

Property (get): `TPDFPrinterStatus PrinterStatus`

This property returns the status of the currently opened printer. See enumeration TPDFPrinterStatus for a comprehensive list of known printer statuses. Multiple statuses may be active at the same time. Therefore, checking for a status should be done using binary operators:

Example: Check whether printer needs to be updated

```
bool bUpdateNeeded = (printer.PrinterStatus & PDFPrinterStatus.ePrinterDriverUpdateNeeded)
!= 0;
```

6.1.65 PrinterStatusMessage

Property (get): `String PrinterStatusMessage`

This property returns a message listing all current statuses of the currently opened printer in a comma-separated list. Returns an empty string if no status is active.

6.1.66 PrintFile

Method: `Boolean PrintFile(String FileName, String PrinterName, String Password, Long FirstPage, Long LastPage)`

Print one file to one printer. This is a "stand-alone" function, the file and printer as provided in the method. This method cannot be combined with other methods such as Open, BeginDocument, or EndDocument.

Parameters:

`FileName` [String] The path and name of the PDF file.

`PrinterName` [String] The name of the printer.

`Password` [String] (optional) The name of the password. A password (owner or user) must be provided when the PDF file is encrypted and has a user password set.

`FirstPage` [Long] (optional) The first page. The default value is `1`.

`LastPage` [Long] (optional) The last page. The default value is `-1` (last page of the document).

Returns:

`True` The document was successfully printed.

`False` Otherwise.

6.1.67 PrintPage

Method: `Boolean PrintPage(Long PageNumber)`

Print a page of the currently opened document. The printer must be selected previously (`OpenPrinter`). This method must be called after `BeginDocument` and before `EndDocument`.

Parameter:

`PageNumber` [Long] This is the number of the page in the PDF file to be printed.

Returns:

`True` The page was successfully printed.

`False` Otherwise.

6.1.68 PrintQuality

Property (get, set): `Integer PrintQuality`
Default: `1 (Printer default)`

Set or get the printing quality. The supported values, from lowest to highest, are:

Value	Description
1	Printer default
DMRES_DRAFT (-1)	Quality = draft
DMRES_LOW (-2)	Quality = low
DMRES_MEDIUM (-3)	Quality = medium
DMRES_HIGH (-4)	Quality = high

6.1.69 ProductVersion

Property (get): String ProductVersion

Get the version of the 3-Heights® PDF Printer API in the format "A.C.D.E".

6.1.70 RenderingMode

Property (get, set): TPDFRenderingMode RenderingMode
Default: eModeFast

Set or get the rendering mode. The supported rendering modes are listed in the enumeration TPDFRendering-Mode. The default and recommended mode is eModeFast.

> **Note:** This property should not be confused with the PrintQuality property. eModeFast is optimized for printing to physical devices and creating a high quality spool file with a small file size.

6.1.71 ReportingLevel

Property (get, set): Integer ReportingLevel
Default: 1

Set or get the reporting level. Available values are:

Value	Description
0	Do not report.

1	Report errors (e.g. file cannot be opened, PDF is corrupted, etc.).
2	Report errors, warnings (e.g. non-embedded font is replaced).
3	Report errors, warnings, information (e.g. page number %d is about to be set).

Errors, warnings, and information are described in the header file `bseerror.h`.

6.1.72 Rotate

Property (get, set): `Integer Rotate`
Default: `0`

Set or get the page's clockwise rotation that is added after the page has been rotated according to the RotateMode. The value is in degrees and must be a multiple of 90.

6.1.73 RotateMode

Property (get, set): `TPDFRotateMode RotateMode`
Default: `eRotateAttribute`

Set or get the rotation of the page. There are four valid values listed under the enumeration TPDFRotateMode.

6.1.74 ScaleXY

Property (get, set): `Float ScaleXY`
Default: `1.0`

After the page has been scaled to fit the paper size, an additional scaling can be specified by using this property. A number less than 1 shrinks the page. A number greater than 1 expands the page. This property can optionally be combined with the FitPage property.

6.1.75 SetCMSEngine

Method: `Boolean SetCMSEngine(String CMSEngine)`

Set the Color Management System (CMS) Engine. The following strings are supported:

"None" The algorithms specified in the PDF reference are used. This results in the maximum possible contrast.

"Neugebauer" The Neugebauer algorithm efficiently converts CMYK to RGB. It does not need any color profiles. The results look similar to conversion using color profiles.

`"lcms"` (default): Use ICC color profiles. Default profiles are used for all unmanaged device color spaces as described in Color profiles.

`FileName` Providing a file name, a configurable version of the Neugebauer algorithm is applied. The coefficients can be defined in the text file. The default Neugebauer coefficients are listed below (Red, Green, Blue; Color):

```
0.996078, 0.996078, 0.996078 ; White
0.000000, 0.686275, 0.937255 ; C
0.925490, 0.149020, 0.560784 ; M
1.000000, 0.949020, 0.066667 ; Y
0.215686, 0.203922, 0.207843 ; K
0.243137, 0.247059, 0.584314 ; CM
0.000000, 0.658824, 0.349020 ; CY
0.066667, 0.176471, 0.215686 ; CK
0.929412, 0.196078, 0.215686 ; MY
0.215686, 0.101961, 0.141176 ; MK
0.200000, 0.196078, 0.125490 ; YK
0.266667, 0.266667, 0.274510 ; CMY
0.133333, 0.098039, 0.160784 ; CMK
0.074510, 0.180392, 0.133333 ; CYK
0.215686, 0.121569, 0.113725 ; MYK
0.125490, 0.121569, 0.121569 ; CMYK
```

The Neugebauer algorithm mixes the colors based on the amount of color and the corresponding weighted coefficient. Altering the values for a pure color specifically changes the result for this pure color.

The color transition remains smooth.

6.1.76 SetLicenseKey

Method: `Boolean SetLicenseKey(String LicenseKey)`

Sets the license key.

6.1.77 SetPaperList

Method: `Boolean SetPaperList(String List)`

Set the list of approved paper sizes used when selecting a paper size automatically (e.g. when property `PaperSize` is set to `-2`).

For example, `SetPaperList("8, 9, 11")` sets the approved paper sizes to A3, A4 and A5.

Parameter:

`List` `[String]` Comma-separated list of paper numbers. Valid paper number values are those listed at the beginning of the strings returned by `GetPaper`.

6.1.78 ShrinkPage

Property (get, set): Boolean ShrinkPage
Default: false

The shrink-page property defines whether the PDF page size should be reduced to fit the paper size. Allowed values are:

Value	Description
True	The page is resized so that both page width and height fit on the printable part of the paper supported by the printer device. The ratio width to height remains unchanged. In contrast to the FitPage property, the resizing is only done if the page size is larger than the paper size.
False	The size of the page remains unchanged. If part of the content is outside the printable area (i.e. close to the border of the page) it will not be printed.

If the FitPage property is True, this property has no effect.

6.1.79 SizeX, SizeY

Property (get, set): Long SizeX
Default: 0

Property (get, set): Long SizeY
Default: 0

Even though paper sizes can be set directly in millimeters using these properties, it is suggested to use the PaperSize property instead.

6.1.80 WaitForJobCompletion

Property (get, set): Boolean WaitForJobCompletion
Default: False

If set to True, the tool waits in EndDocument until the spooler reports that the job has been completed. This ensures that temporarily installed fonts are not removed until the print job has been completed. In conjunction with virtual printer drivers or when printing to a network printer (especially to Windows 2012 print servers), this switch is recommended.

If the input file contains embedded fonts, their use is not disabled. Printing is done locally, then the wait function is enabled by default. With the property, the wait function can be enabled unconditionally.

6.1.81 WatermarkAlignRight

Property (get, set): Boolean WatermarkAlignRight
Default: False

Set or get whether added watermark text is aligned right or left of the given postition.

6.1.82 WatermarkAngle

Property (get, set): `Single WatermarkAngle`
Default: `0`

Set or get the angle of the watermark in radians. To transform a degree value to a radian value, multiply the degree value by factor 2π / 360.

6.1.83 WatermarkBold

Property (get, set): `Boolean WatermarkBold`
Default: `False`

Set or get if a bold font is used.

6.1.84 WatermarkColor

Property (get, set): `Long WatermarkColor`
Default: `0 (Black)`

Set or get the color of the watermark. The color is in RGB. The value is calculated as:

Color = red + 256 × green + 256^2 × blue,

where red, green and blue are values from 0-255.

6.1.85 WatermarkFileName

[Deprecated] Property (get, set): `String WatermarkFileName`
Default: `""`

Use AddWatermarkImage instead. This property can be used to embed a raster image or PDF as watermark logo. The supported image types are BMP, GIF, JPEG, PNG, and TIFF. Transparency and alpha channels are not supported for raster images.

6.1.86 WatermarkFontName

Property (get, set): `String WatermarkFontName`
Default: `""`

Set or get the font name of the watermark text.

6.1.87 WatermarkFontSize

Property (get, set): `Single WatermarkFontSize`
Default: `12`

Set or get the size of the font of the watermark text.

6.1.88 WatermarkInBackground

Property (get, set): `Boolean WatermarkInBackground`
Default: `False`

Set or get whether the watermark should be placed in the foreground or background layer.

6.1.89 WatermarkItalic

Property (get, set): `Boolean WatermarkItalic`
Default: `False`

Set or get whether the font property italic is turned on or off.

6.1.90 WatermarkOutline

Property (get, set): `Boolean WatermarkOutline`
Default: `False`

Set or get whether the font property outline is turned on or off. Outline refers to only stroking the text, without filling it.

6.1.91 WatermarkScale

Property (get, set): `Single WatermarkScale`
Default: `1`

Set or get the scaling factor to scale raster images. A value of 1 equals 100%, i.e. no scaling.

6.1.92 WatermarkText

[Deprecated] Property (get, set): `String WatermarkText`

Deprecated, use AddWatermarkText instead.

6.1.93 WatermarkXPos, WatermarkYPos

Property (get, set): Single WatermarkXPos
Default: 0 (left)

Property (get, set): Single WatermarkYPos
Default: 0 (top)

Use WaterkmarkXPos to set or get the horizontal position of the watermark. 0 is on the left side.

Use WaterkmarkYPos to set or get the vertical position of the watermark. 0 is on the top of the page.

The units of the coordinate system are 1/72 inch on the PDF or image printed[8].

6.2 Enumeration

Note: Depending on the interface, enumerations may have TPDF as prefix (COM, C), PDF as prefix (.NET), or no prefix at all (Java).

6.2.1 TPDFErrorCode Enumeration

All TPDFErrorCode enumerations start with a prefix, such as PDF_, followed by a single letter which is one of S, E, W or I, an underscore, and a descriptive text.

The single letter gives an indication of the severity of the error. These are: Success, Error, Warning, and Information. In general, an error is returned if an operation could not be completed. A warning is returned if the operation was completed, but problems occurred in the process.

A list of all error codes is available in the C API header file bseerror.h, the javadoc documentation of com.pdftools.NativeLibrary.ERRORCODE, and the .NET documentation of Pdftools.Pdf.PDFErrorCode. Note that only a few are relevant for the 3-Heights® PDF Printer API, most of which are listed here:

TPDFErrorCode table

TPDFErrorCode	Description
PDF_S_SUCCESS	The operation was completed successfully.
LIC_E_NOTINIT, ... LIC_E_LEVEL	Various license management related errors.
PDF_E_FILEOPEN	Failed to open the file.
PDF_E_FILECREATE	Failed to create the file.
PDF_E_PASSWORD	The authentication failed due to a wrong password.

[8] A PDF page of size "A4" is 595 by 842 points. A page of size "Letter" is 612 by 792 points.

TPDFErrorCode table

PDF_E_UNKSECHANDLER	The file uses a proprietary security handler, e.g. for a proprietary digital rights management (DRM) system.
PDF_E_COLLECTION	The input file is a PDF collection without an initial document
PDF_E_XFANEEDSRENDERING	The file contains unrendered XFA form fields, i.e. the file is an XFA and not a PDF file. The XFA (XML Forms Architecture) specification is referenced as an external document to ISO 32'000-1 (PDF 1.7) and has not yet been standardized by ISO. Technically spoken, an XFA form is included as a resource in a shell PDF. The PDF's page content is generated dynamically from the XFA data, which is a complex, non-standardized process. For this reason, XFA is forbidden by the ISO Standards ISO 19'005-2 (PDF/A-2) and ISO 32'000-2 (PDF 2.0) and newer.

6.2.2 TPDFPrinterStatus Enumeration

The printer status indicates any issues with the queried printer driver.

TPDFPrinterStatus table

TPDFPrinterStatus	Description
ePrinterPaused	The printer is paused.
ePrinterError	The printer is in an error state.
ePrinterPaperJam	Paper is jammed in the printer.
ePrinterPaperOut	The printer is out of paper.
ePrinterPaperProblem	The printer has a paper problem.
ePrinterOffline	The printer is offline.
ePrinterOutputBinFull	The printer's output bin is full.
ePrinterNotAvailable	The printer is not available for printing.
ePrinterNoToner	The printer is out of toner.
ePrinterOutOfMemory	The printer has run out of memory.
ePrinterDoorOpen	The printer door is open.
ePrinterDriverUpdateNeeded	The printer driver needs to be updated.

6.2.3 TPDFRendererOption Enumeration

Renderer options are set using the property Options. To combine multiple options, use a bitwise OR operator. To disable an option, use the bitwise AND NOT operators.

Example: Visual Basic

Enable or disable an option, and leave all other options untouched:

```
' Enable high quality rendering (anti-aliasing)
.Options = .Options OR eOptionHighQuality
' Disable high quality rendering (anti-aliasing)
.Options = .Options AND NOT eOptionHighQuality
```

Example: C/C++

```
int iOptions = PDFPrnGetOptions (hPrinter);
// Enable high quality rendering (anti-aliasing)
PDFPrnSetOptions (hPrinter, iOptions | eOptionHighQuality);
// Disable high quality rendering (anti-aliasing)
PDFPrnSetOptions (hPrinter, iOptions & ~eOptionHighQuality);
```

The following list includes renderer options that are relevant for the 3-Heights® PDF Printer API. There are more enumerations available, but they are unrelated to this API.

TPDFRendererOption Table

TPDFRendererOption	
eOptionAutoAccurateMode	Detect content that cannot be rendered using RenderingMode eModeFast (GDI) and switch to eModeAccurate (GDI+) automatically, e.g. to render transparent tiling patterns. It does not have any effect if eModeAccurate was already set.
eOptionBanding	(default) The data is sent in small chunks. This is mainly for older printers with limited memory.
eOptionBilinear	A bilinear image filter is applied to images to improve the image quality. This option cannot be combined with eOptionBicubic.
eOptionBitmap	The pages are rendered in a bitmap, which then is sent to the device.
eOptionDisableAnnots	If this option is set, annotations are not printed.
eOptionDisableBPC	If this option is set, then the black point compensation feature is disabled when converting colors e.g. from CMYK to RGB.
eOptionDisableBuffer	Disable the transparency backbuffer.
eOptionDisableContent	If this option is set, then only form fields and annotations are printed without the underlying page content.
eOptionDisableFilter	Disable image filtering. Images are scaled using the nearest-neighbor algorithm, which improves performance at the cost of rendering quality.
eOptionDisablePatterns	Disable patterns.
eOptionDisablePS	Disable direct PostScript injection.

TPDFRendererOption Table

eOptionDrawPopups	Print pop-up windows of annotations, such as sticky notes.
eOptionFillStroke	If this option is set, then strokes are converted to filled paths.
eOptionJPEG	If a printer supports JPEG compression, then the pages can be sent with JPEG compression to reduce the size of the spool file.
eOptionHighQuality	(default) Anti-aliasing for text and path objects and filtering of image objects can be turned off and on with this option.
eOptionNoEmbedded	Do not use embedded fonts. Instead fonts from the operating system's font directory are used (%Systemroot%\fonts).
eOptionOutlines	Convert fonts into vector graphics.
eOptionPreInstalled	Replace embedded fonts with a pre-installed font if the same font is already installed on the OS.
eOptionPNG	If a printer supports PNG compression, then the pages can be sent with PNG compression to reduce the size of the spool file.
eOptionPrint	(default) Print the document as it was intended for printing. Otherwise, the document is printed as it is shown in an interactive viewer. For example, this has an effect on which annotations are visible.
eOptionPrintOnlySig	Print the digital signature appearance only (without any status appearances, e.g. valid or invalid).
eOptionDoNotPrintSig	Do not print digital signature appearances.
eOptionPSLevel2	Use PostScript language level 2 for printers that do not provide information on their supported language level. Without this option, language level 3 is used.
eOptionTransparency	Deprecated option that has no effect.
eOptionTrueType	(default) CFF and Type1 fonts are converted to True Type fonts. This option overrules option eOptionType1.
eOptionType1	CFF fonts are converted to Type1 fonts.
eOptionUseFastImages	Always print images in fast mode. This should help resolving performance issues with complex images and image masks of documents that are to be printed in accurate mode.

TPDFRendererOption Table

eOptionUseUnicodes	Use Unicode code points instead of glyph IDs for embedded fonts. This option is to create spool files with text that is optimized for post-processing. This guarantees that text using embedded fonts remains extractable. In addition, invisible (OCR) text is printed as white text to the background of the page. While this has no effect on the visual appearance of the page, this preserves the extractable text, e.g. when printing scanned documents.
eOptionWindows9x	Run in Windows 9x compatibility mode. Setting this option can resolve issues with nested transformation matrices, which are not supported by all devices.

6.2.4 TPDFRenderingMode Enumeration

TPDFLayoutMode table

TPDFRenderingMode	
eModeAccurate	The accurate mode is intended for virtual printers such as a TIFF printer. It uses the Windows GDI+ for rendering. This mode allows for image filtering, sub-pixel rendering, and anti-aliasing. It should not be applied for physical devices, such as a laser printer, due to the fact that those devices do not support the above features. Using the accurate mode creates generally larger spool files than the fast mode.
eModeDirect	This mode is deprecated.
eModeFast	The fast mode is the recommended mode for printing to any physical printer device such as a laser printer, or ink jet printer. It uses the Windows GDI for rendering. This mode is generally faster and creates smaller spool files than the accurate mode. Use this mode for high resolution (600 DPI).

6.2.5 TPDFRotateMode Enumeration

TPDFViewerOption table

TPDFRotateMode	
eRotateAttribute	Set the rotation to the viewing rotation attribute of the PDF page, i.e. rendering the page with the same rotation as it is displayed in a PDF viewer.
eRotatePortrait	Rotate page to portrait.
eRotateLandscape	Rotate page to landscape.
eRotateNone	Process the page as it is saved in the PDF file.

7 Troubleshooting

7.1 General

7.1.1 No output

Check the return codes of:

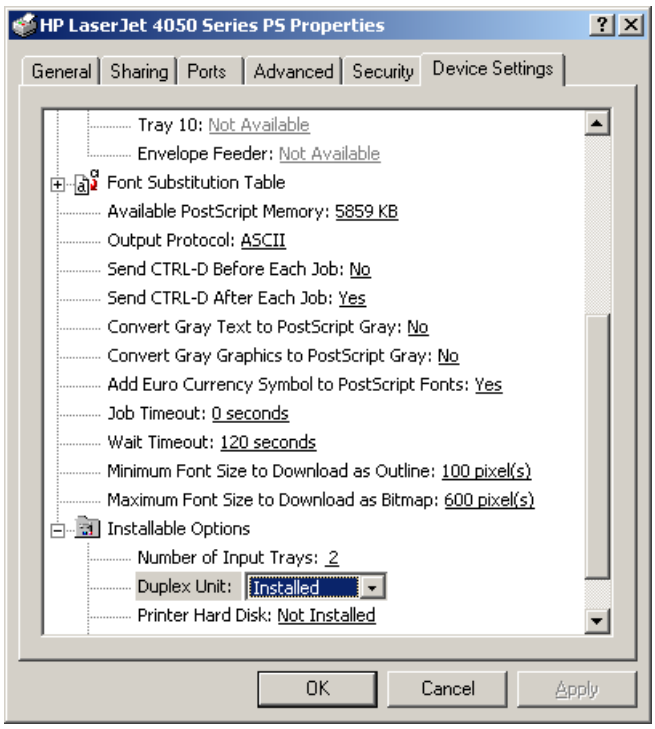
- `OpenPrinter`: If false, the printer driver could not be found. Verify that the name of the printer is written correctly, the printer is installed, and the user has permission rights to access it. Missing permission rights is a common cause in a network or web application for `OpenPrinter` to fail.
- `BeginDocument`: If false, the connection to the physical printer device failed. Verify that the printer device is connected to the networks and turned on. This method also returns false if OpenPrinter failed previously.
- `Open`: If false, the PDF document could not be opened. Check for spelling errors. Verify the PDF document is not corrupted (or test with another document). Also check for permission rights.
- `PrintPage`: If false, the selected page could not be printed. Check page range.
- `PrintFile`: This method does all the four steps above (plus `Close`, `EndDocument`, `ClosePrinter`) in one. Therefore, if `PrintFile` returns false, any of the above steps could have failed.
- Verify the property `Output` is not set. Otherwise, the output is redirected to a file.

7.1.2 Blank output

If you are printing a very complex document or a document with very large embedded raster images, it may help to reduce the resolution of the printer (in the printer's properties), e.g. from 1200 to 600 DPI.

7.1.3 Duplex mode is not listed or does not work

If the duplex mode is not listed, check if the printer has an option that needs to be installed to allow duplex printing (see screenshot).



Go to "Start" →"Settings" →"Printer" → right-click your printer →"Properties" →"Device Properties". Look for options like "Installable Options" →"Duplex Unit".

If you can print duplex using other Windows applications, try using the value 2 or 3 as parameter.

Alternatively to the `Duplex` property, the duplex mode can be defined in the printer's properties. See Using the device mode.

7.1.4 Page does not fit the paper

Setting the `FitPage` property scales the page to fit the paper size. This property should be set when the dimensions of the PDF and the dimension of the paper size are different.

Optionally, use `Center` = `True` to center the page vertically and horizontally. The position can additionally be moved using the `OffsetX`, `OffsetY` properties.

7.1.5 Orientation

Every page in a PDF document can have a separate rotation value. For example, a page that visually appears as a landscape can actually be a 90° rotated portrait. When sending such a page to the printer, it is treated according to the `RotateMode` property, which is by default set to `eRotateAttribute` and thus is sent the way it is viewed — as landscape.

Then the next property comes into play: `Orientation`. While `RotateMode` was for the page, `Orientation` is for the paper. The default of this property is `Orientation` = `-1`, which means the printer defaults are used. `-2` means the automatic mode is used. This sets the orientation so that the page best fits the paper. The values `1` and `2` force the orientation to portrait and landscape.

Example: If you would like a page that appears as landscape to be printed as landscape (and filling the paper), use the following settings:

```
FitPage = True
```

```
Orientation = -2
```

Example: If you would like a landscape to be printed as portrait, and only fill half of the paper, use a setting like this:

```
FitPage = True
Orientation = 1
```

7.1.6 Printer settings or device mode ignored

Some printer drivers ignore some properties set at the 3-Heights® PDF Printer API. They usually respect the settings made in the printing properties dialog only. Since printing properties set in the dialog are stored in the device mode, the best way to cope with such printer drivers is to create and use a device mode as described in Using the device mode.

Furthermore, certain settings applied to the device mode behave differently on local and network printers. It does not matter whether using device mode functions of the 3-Heights® PDF Printer API, or adjusting the defaults in the printer itself.

A very basic setting in the device mode, such as “print as landscape”, should always work, whereas a more complex setting such as “print multiple pages on 1 paper” may fail on a local printer, but work on a network printer. This is due to the nature of how the printing system works on Windows. A detailed explanation is not provided here, but a workaround to this type of issue is normally using the EMF mode (as a side-effect, this simulates a network environment even for local printers).

7.1.7 Printer ignores device mode configuration

If the datatype is set to either RAW or EMF, some printer drivers may completely ignore any configuration that was performed using the device mode structureas described in Using the device mode.

To avoid this, delegate the decision on the datatype to use to the printer driver using the `DataType` property.

7.1.8 Black is not printed completely black

Sometimes black color is not printed completely black. This is due to color transformations between different color spaces.

Black point compensation allows for higher contrast of the black color. It is applied automatically if no color profile is specified (i.e. no color profiles are available in the subdirectory `Icc` or an appropriate color profile is not found on the system). In this situation, the conversion is done algorithmically using Neugebauer and black point compensation.

7.2 Spool file size

If the size of spool files should be reduced, the following points can be considered:

- Rendering mode
- Printer driver

- Network environment, RAW/EMF mode
- Resolution

7.2.1 Rendering mode

The 3-Heights® PDF Printer API supports two rendering modes: Fast (default) and Accurate. The Fast mode uses the GDI, whereas the Accurate mode uses the GDI+. In the Accurate mode, there are several filters available. These filters are intended for low resolution devices such as a monitor or a raster image. On a 600 DPI resolution printer, anti-aliasing has almost no visual impact. In fact, most printers do not even support anti-aliasing. Therefore it is generally suggested to use the Fast mode. However, there are certain documents that print quite differently using GDI or GDI+ for other reasons.

7.2.2 Printer driver

Most printer devices understand more than one printer language. For example, most HP printers support different types of PCL (Printer Command Language) such as PCL 5, PCL 5e or PCL 6, and in addition, PostScript. There are also printer devices that only support one printer language. It is usually best (and also suggested by printer manufacturers) to use the printer driver that works best. If PostScript yields large spool files or has rendering issues, try a PCL printer driver or vice versa.

The smallest spool sizes can be achieved by using either PostScript or PCL 6. This is heavily dependent on the PDF input file.

7.2.3 PostScript injection

The reason why different applications can create spool files of very different sizes of the same PDF document is the way the spool file is created.

PostScript is generated using the `PScript5.dll`. There are different plugins, which are printer driver-dependent. These plugins are `.psd` file. For example, this can be something like `hp4050.psd`.

A part of the created spool file uses Document Structuring Conventions (DSC) language. These commands are printer driver-dependent and may look like this:

```
%%Title: input.pdf
%%Creator: PScript5.dll Version 5.2.2
%%CreationDate: 5/23/2005 11:40:2
%%For: pre
%%BoundingBox: (atend)
%%DocumentNeededResources: (atend)
%%DocumentSuppliedResources: (atend)
%%DocumentData: Clean7Bit
%%TargetDevice: (HP LaserJet 4050 Series) (2014.108) 1
%%LanguageLevel: 2
%%EndComments
```

The DSC is used to define the page settings and all printer driver-dependent properties.

In between the DCS comments, there are the actual PostScript commands (all the parts that do not start with %%), which provide all the information about the content of the page.

An application that is printing a spool file can first ask the GDI whether the type is PostScript. If the GDI says yes, then there is a “pass through mode”, which can be used to provide the PostScript commands directly and let the

printer driver only take care of the DSC. This called direct PostScript injection. Some printer drivers do not support this. In such cases, it should be turned off.

7.2.4 Resolution

Most printers support different resolutions, such as 300 DPI, 600 DPI, 1200 DPI, etc. Depending on the printer language and the document, the resolution influences the spool size. For printer devices that require raster graphics to be provided uncompressed and at device resolution, the size of an image at 1200 DPI is 16 times the size as at 300 DPI.

The `PrintQuality` property can be used to set the printing resolution.

The property `MaxDPI` allows to limit the resolution of pre-rendered images.

7.2.5 Multiple copies

In combination with `PrintFile`, multiple copies can be printed using the `Copies` property. The `Collate` property defines if the output is printed by page or by document. If a printer does not support collate, the collate mode can be simulated by using `CopyMode` or by simply printing a page or document multiple times. However, these two alternatives create the pages multiple times in the spool files and increase its file size.

7.3 Printing in a network environment

It is preferable to not send large spool files over the network. To handle this, there are two similar approaches.

- Print the PDF at its destination: Usually a PDF is much smaller than a spool file. Therefore, it makes sense to not print the PDF first and send a large spool file over the network, but instead send the PDF over the network and print it at its destination.
- Use EMF mode instead of RAW: By using the EMF (Windows Embedded Metafile) mode, the document is sent as EMF over the network and spooled at its destination. This has the advantage of sending much less data over the network because the RAW spool file (e.g. PCL or PS) is created locally. The downside is possible issues with printer driver at the remote site.

Use network shared printers with caution: Using shared printer resources in the Windows operating system always involves that printer drivers are transferred from the printer server to the client computer. The shared printer resource should be mapped as a user with administrator rights to prevent from a failure of the printing application to open the printer connection.

Check permissions: The user of a printing application must at least have the “print” permission to use the referred printer object.

Use the `WaitForJobCompletion` property to prevent issues printing embedded fonts. This feature ensures that temporarily installed fonts are not removed until the print job has been completed.

Retry calls: Calls to the Windows operating system may fail while printing to a remote printer, even in a reliable network. An application program may want to repeat calls to the API until they succeed. This is allowed for calls to `OpenPrinter` and `BeginDocument` as these functions have been designed for this purpose.

7.4 Multithreaded printing

The 3-Heights® PDF Printer API is fully thread-safe as long as one `Printer` is used in one thread at the time only. However, not all printer drivers are thread-safe. To cope with this, it is recommended to synchronize calls to `BeginDocument`. Notably, the error message “Invalid Handle” after `PrintPage` is an indication for multithreading issues in the printer driver.

7.5 Font and text issues

1. For issues with text using non-embedded fonts:
   1. Ensure the required fonts are available on the system (see Fonts).
   2. See Handle non-embedded fonts.
2. For issues with text using embedded fonts:
   1. Ensure embedded fonts are used (i.e. `eOptionNoEmbedded` is not set).
   2. Ensure the two system environment variables, TEMP and TMP, exist and point to an existing directory. Not setting the environment variables often results in errors for service applications that run under a user that has no temporary directory and thus cannot install fonts. See also Installation and deployment.
   3. If you are using a local printer, ensure `DataType` is set to `"raw"`.
   4. If you are using a remote printer, see Printing in a network environment.
   5. See Handle embedded fonts.
3. If you are using an older printer driver, try `eOptionWindows9x` or install a newer printer driver.
4. Try a different type of printer driver, e.g. PCL 6 instead of PS or vice versa.

7.5.1 Handle non-embedded fonts

Font replacement strategy

This section describes how the rendering engine handles fonts. It is rather technical and it is not required to be understood to use the software.

The following steps are performed sequentially when searching for a font. If a font is found, the search is stopped; otherwise, the next step is performed.

1. If the font is not embedded or `eOptionPreInstalled` is set:
   a. If the font name appears in the `[replace]` section in the configuration file `fonts.ini`, the name is replaced and looked up in the installed font collection.
   b. If it is a standard font[9] it is replaced by the equivalent TrueType font name and it is looked up in the installed font collection.
   c. If the font name appears in the `[fonts]` section in the configuration file `fonts.ini`, the name is replaced and looked up in the installed font collection.
   d. If the font has “Italic” or “Bold” in its name, the font without these styles is looked up in the installed font collection.
2. If a font name is looked up in the installed font collection, then the name comparison is performed as follows:
   a. PostScript name.
   b. TrueType name without blanks (a missing style is interpreted as “Regular” or “Normal”).
   c. TrueType name without modifications.
3. If the font is embedded, it is converted to a Windows-compatible font and temporarily installed. If `eOptionNoEmbedded` is used, then the glyphs of the fonts are converted to either bitmaps or outlines[10]. If `eOptionOutlines` is used, then the glyphs are converted to outlines only.
4. If the font is not embedded and the Unicode code points are available, then the nearest font from the installed font collection is tailored to the metrics of the font.

[9] e.g. Times-Roman, Helvetica, Courier
[10] The outline of a glyph is a vector graphic without any reference to the original font program.

7.5.2 Handle embedded fonts

The following list provides possible work-arounds if text is printed incorrectly. Options should be tried in ascending order.

1. Using the option `eOptionNoEmbedded` inhibits all embedded fonts from being used in the spool file and the printer hardware. Instead, the glyphs are converted to either bitmaps or outlines. Using the option `eOptionOutlines` at the same time the conversion is restricted to outlines.
2. Using the option `eOptionPreInstalled` inhibits embedded fonts which have the same name as the corresponding installed font from being used. This option can also be used to reduce the number of fonts in a spool file if the printer hardware memory capacity is limited.
3. Pre-render the page in a bitmap and send the pre-rendered image to the printer (`eOptionBitmap`). This results in large spool files.

7.6 Unsupported PDF features

The 3-Heights® Rendering Engine supports transparency functions such as a number of blend modes and isolated and non-isolated transparency groups, but not transparency in general.

Filling geometric figures with tiling and shading patterns may fail in some cases.

8 Version history

Some of the documented changes below may be preceded by a marker that specifies the interface technologies the change applies to. For example, [C, Java] applies to the C and the Java interface.

8.1 Changes in versions 6.19–6.27

- **Update** license agreement to version 2.9

8.2 Changes in versions 6.13–6.18

No functional changes.

8.3 Changes in versions 6.1–6.12

- **Improved** search algorithm for installed fonts: User fonts under Windows are now also taken into account.
- [Java] **Changed** minimal supported Java language version to 7 [previously 6].
- [.NET] **New** availability of this product as NuGet package for Windows, macOS and Linux.
- [.NET] **New** support for .NET Core versions 1.0 and higher. The support is restricted to a subset of the operating systems supported by .NET Core, see Operating systems.
- [.NET] **Changed** platform support for NuGet packages: The platform "AnyCPU" is now supported for .NET Framework projects.

8.4 Changes in version 5

- **New** additional supported operating system: Windows Server 2019.
- **Changed** behavior when reading a TIFF. The value `Relative` from tag `ResolutionUnit` is now interpreted as `Inch`.
- **Changed** opening of remote printers to not use locally cached data but retrieve effective data from printer.

8.5 Changes in version 4.12

- **Improved** the performance when printing image masks in accurate mode.
- **Improved** reading and recovery of corrupt TIFF images.
- **New** HTTP proxy setting in the GUI license manager.

8.6 Changes in version 4.11

- **New** support for reading PDF 2.0 documents.
- **New** Property `MaxDPI` to set the maximum DPI that is used for images sent to the printer. Useful for printers with low memory capacity.
- **New** Property `PrinterStatus` queries the current status of the printer.
- **New** Property `PrinterStatusMessage` provides a human readable message listing all active statuses.

8.7 Changes in version 4.10

- **Improved** robustness against corrupt input PDF documents.
- **Improved** annotation appearance generation for polyline, squiggly, and stamp annotations.
- **Removed** the font `ZapfDingbats.ttf` from the product kit as it is not required anymore.
- [C] **Clarified** Error handling of `TPdfStreamDescriptor` functions.
- **Changed** method `SetDataType` to accept input values `"null"` or `""`, to delegate setting of data type to the printer driver.

8.8 Changes in version 4.9

- **Improved** support for and robustness against corrupt input PDF documents.
- **Improved** repair of embedded font programs that are corrupt.
- **New** support for OpenType font collections in installed font collection.
- **Improved** metadata generation for standard PDF properties.
- [C] **Changed** return value `pfGetLength` of `TPDFStreamDescriptor` to `pos_t`[11].
- **Changed** behavior of enumeration value `eOptionBitmap`: New Rendering Engine 2.0 is used.
- [.NET, Java] **New** method `OpenStream()`.
- **Changed** property `DataType`: Datatype can be set to `Nothing` to inherit datatype from driver.

8.9 Changes in version 4.8

- **Improved** creation of annotation appearances to use less memory and processing time.
- **Added** repair functionality for TrueType font programs whose glyphs are not ordered correctly.
- [.NET, C, COM, Java] **New** property `Rotate` to rotate pages.
- [.NET, C, COM, Java] **New** property `WatermarkinBackground` to move watermark to background layer.
- [.NET, C, COM, Java] **New** property `WatermarkAlignRight` to change text alignment of watermark.
- [.NET, C, COM, Java] **New** property `ProductVersion` to identify the product version.
- [.NET] **Deprecated** method `GetLicenseIsValid`.
- [.NET] **New** property `LicenseIsValid`.

[11] This has no effect on neither the .NET, Java, nor COM API

9 Licensing, copyright, and contact

Pdftools (PDF Tools AG) is a world leader in PDF software, delivering reliable PDF products to international customers in all market segments.

Pdftools provides server-based software products designed specifically for developers, integrators, consultants, customizing specialists, and IT departments. Thousands of companies worldwide use our products directly and hundreds of thousands of users benefit from the technology indirectly via a global network of OEM partners. The tools can be easily embedded into application programs and are available for a multitude of operating system platforms.

Contact

PDF Tools AG
Brown-Boveri-Strasse 5
8050 Zürich
Switzerland
https://www.pdf-tools.com
pdfsales@pdf-tools.com

A Default values

A.1 Duplex modes

1	Simplex
2	Vertical duplex
3	Horizontal duplex

A.2 Paper bins

1	Upper
2	Lower
3	Middle
4	Manual
5	Envelope
6	Envelope Manual
7	Auto
8	Tractor
9	Small FMT
10	Large FMT
11	Large capacity
12	undef.
13	undef.
14	Cassette
15	From Source

A.3 Paper sizes

1	Letter 8 1/2 x 11 in
2	Letter Small 8 1/2 x 11 in
3	Tabloid 11 x 17 in
4	Ledger 17 x 11 in
5	Legal 8 1/2 x 14 in
6	Statement 5 1/2 x 8 1/2 in
7	Executive 7 1/4 x 10 1/2 in
8	A3 297 x 420 mm
9	A3 297 x 420 mm
10	A4 Small 210 x 297 mm
11	A5 148 x 210 mm
12	B4 (JIS) 250 x 354
13	B5(JIS)182x257mm
14	Folio 8 1/2 x 13 in
15	Quarto 215 x 275 mm
16	10x14 in
17	11x17 in
18	Note 8 1/2 x 11 in
19	Envelope # 9 3 7/8 x 8 7/8
20	Envelope # 10 4 1/8 x 9 1/2
21	Envelope # 11 4 1/2 x 10 3/8
22	Envelope # 12 4 \276 x 11
23	Envelope # 14 5 x 11 1/2
24	C size sheet
25	D size sheet
26	E size sheet
27	Envelope DL 110 x 220mm

28	Envelope C5 162 x 229 mm
29	Envelope C3 324 x 458 mm
30	Envelope C4 229 x 324 mm
31	Envelope C6 114 x 162 mm
32	Envelope C65 114 x 229 mm
33	Envelope B4 250 x 353 mm
34	Envelope B5 176 x 250 mm
35	Envelope B6 176 x 125 mm
36	Envelope 110 x 230 mm
37	Envelope Monarch 3.875 x 7.5 in
38	63/4Envelope35/8x61/2in
39	USStdFanfold147/8x11in
40	German Std Fanfold 8 1/2 x 12 in
41	German Legal Fanfold 8 1/2 x 13 in
42	B4 (ISO) 250 x 353 mm
43	Japanese Postcard 100 x 148 mm
44	9 x 11 in
45	10 x 11 in
46	15 x 11 in
47	Envelope Invite 220 x 220 mm
48	RESERVED—DO NOT USE
49	RESERVED—DO NOT USE
50	Letter Extra 9 \275 x 12 in
51	Legal Extra 9 \275 x 15 in
52	Tabloid Extra 11.69 x 18 in
53	A4 Extra 9.27 x 12.69 in
54	Letter Transverse 8 \275 x 11

55	A4 Transverse 210 x 297 mm
56	Letter Extra Transverse 9\275
57	SuperA/SuperA/A4 227 x 356
58	SuperB/SuperB/A3 305 x 487
59	Letter Plus 8.5 x 12.69 in
60	A4 Plus 210 x 330 mm
61	A5 Transverse 148 x 210 mm
62	B5 (JIS) Transverse 182 x 257 mm
63	A3 Extra 322 x 445 mm
64	A5 Extra 174 x 235 mm
65	B5 (ISO) Extra 201 x 276 mm
66	A2 420 x 594 mm
67	A3 Transverse 297 x 420 mm
68	A3 Extra Transverse 322 x 445 mm
69	Japanese Double Postcard 200 x 148 mm
70	A6 105 x 148 mm
71	Japanese Envelope Kaku # 2
72	Japanese Envelope Kaku # 3
73	Japanese Envelope Chou # 3
74	Japanese Envelope Chou # 4
75	Letter Rotated 11 x 8 1/2 in
76	A3 Rotated 420 x 297 mm
77	A4 Rotated 297 x 210 mm
78	A5 Rotated 210 x 148 mm
79	B4 (JIS) Rotated 364 x 257 mm
80	B5 (JIS) Rotated 257 x 182 mm
81	Japanese Postcard Rotated 148 x 100 mm

82	Double Japanese Postcard Rotated 148 x 200 mm
83	A6 Rotated 148 x 105 mm
84	Japanese Envelope Kaku # 2 Rotated
85	Japanese Envelope Kaku # 3 Rotated
86	Japanese Envelope Chou # 3 Rotated
87	Japanese Envelope Chou # 4 Rotated 88B6(JIS)128x182mm
89	B6 (JIS) Rotated 182 x 128 mm
90	12x11in
91	Japanese Envelope You # 4
92	Japanese Envelope You # 4 Rotated
93	PRC 16K 146 x 215 mm
94	PRC 32K 97 x 151 mm
95	PRC 32K(Big) 97 x 151 mm
96	PRC Envelope # 1 102 x 165 mm
97	PRC Envelope # 2 102 x 176 mm
98	PRC Envelope # 3 125 x 176 mm
99	PRC Envelope # 4 110 x 208 mm
100	PRC Envelope # 5 110 x 220 mm
101	PRC Envelope # 6 120 x 230 mm
102	PRC Envelope # 7 160 x 230 mm
103	PRC Envelope # 8 120 x 309 mm
104	PRC Envelope # 9 229 x 324 mm
105	PRC Envelope # 10 324 x 458 mm
106	PRC 16K Rotated
107	PRC 32K Rotated
108	PRC 32K(Big) Rotated
109	PRC Envelope # 1 Rotated 165 x 102 mm

110	PRC Envelope # 2 Rotated 176 x 102 mm
111	PRC Envelope # 3 Rotated 176 x 125 mm
112	PRC Envelope # 4 Rotated 208 x 110 mm
113	PRC Envelope # 5 Rotated 220 x 110 mm
114	PRC Envelope # 6 Rotated 230 x 120 mm
115	PRC Envelope # 7 Rotated 230 x 160 mm
116	PRC Envelope # 8 Rotated 309 x 120 mm
117	PRC Envelope # 9 Rotated 324 x 229 mm
118	PRC Envelope # 10 Rotated 458 x 324 mm