User Manual



3-Heights™ PDF Optimizer API

Version 4.8

PDF-TOOLS.COM
Premium PDF Technology

Contents

1 Introduction 4
1.1 Description 4
1.2 Functions 4
1.2.1 Features 4
1.2.2 Formats 5
1.2.3 Compliance 5
1.3 Interfaces 6
1.4 Operating Systems 6

2 Installation and Deployment 7
2.1 Windows 7
2.2 Unix 7
2.2.1 All Unix Platforms 8
2.2.2 macOS 8
2.3 Interfaces 9
2.3.1 Development 9
2.3.2 Deployment 11
2.4 Interface Specific Installation Steps 11
2.4.1 COM Interface 11
2.4.2 Java Interface 12
2.4.3 .NET Interface 12
2.4.4 C Interface 13
2.5 Uninstall, Install a New Version 13
2.6 Color Profiles 13
2.6.1 Default Color Profiles 13
2.6.2 Get Other Color Profiles 14
2.7 Note about the Evaluation License 14

3 License Management 15
3.1 Graphical License Manager Tool 15
3.1.1 List all installed license keys 15
3.1.2 Add and delete license keys 15
3.1.3 Display the properties of a license 15
3.1.4 Select between different license keys for a single product 15
3.2 Command Line License Manager Tool 16
3.3 License Key Storage 16
3.3.1 Windows 16
3.3.2 macOS 16
3.3.3 Unix/Linux 17

4 Programming Interfaces 18
4.1 Visual Basic 6 18
4.2 ASP - VBScript 19
4.3 .NET 19
4.3.1 Visual Basic 20
4.3.2 C# 21
4.3.3 Deployment 22
Detailed description 22
Simplified description 22

4.3.4 Troubleshooting: TypeInitializationException . . . 22
Troubleshooting: DllNotFoundException . . . 22
Troubleshooting: BadImageFormatException . . . 23

5 User's Manual . . . 24
5.1 Overview of the API . . . 24
5.2 How to Optimize PDF Documents . . . 24
5.2.1 Identify Target Application Area . . . 24
Web . . . 24
Printing . . . 25
Archiving . . . 25
Scanned Documents . . . 26
Special Requirements . . . 26
5.2.2 Using Optimization Profiles . . . 26
5.3 Optimizing Images . . . 26
5.3.1 Supported Image Compression Types . . . 26
No Compression (Raw) . . . 27
DCT (JPEG) . . . 27
Flate (ZIP) . . . 27
LZW . . . 28
CCITT Fax Group 3 and 4 . . . 28
JBIG2 . . . 28
JPEG2000 . . . 29
5.3.2 Relevant Factors for the File Size . . . 29
5.3.3 Provided Features for Optimizing Images . . . 30
5.3.4 Mixed Raster Content (MRC) Optimization for Images . . . 32
Phase 1: Cutting out Pictures . . . 32
Phase 2: Separation into Layers . . . 33
Phase 3: Reconstruction . . . 33
5.4 Optimizing Fonts . . . 33
5.5 Extracting Resources . . . 34

6 Reference Manual . . . 35
6.1 `PDFOptimizer` Interface . . . 35
6.1.1 `BitonalCompression` . . . 35
6.1.2 `BitonalCompressions` . . . 35
6.1.3 `BitonalResolutionDPI` . . . 36
6.1.4 `BitonalThresholdDPI` . . . 36
6.1.5 `ClipImages` . . . 36
6.1.6 `Close` . . . 36
6.1.7 `ColorCompression` . . . 37
6.1.8 `ColorConversion` . . . 37
6.1.9 `ColorResolutionDPI` . . . 37
6.1.10 `ColorThresholdDPI` . . . 37
6.1.11 `ContinuousCompressions` . . . 38
6.1.12 `CompressionQuality` . . . 38
6.1.13 `ConvertToCFF` . . . 38
6.1.14 `DitheringMode` . . . 38
6.1.15 `ErrorCode` . . . 39
6.1.16 `ExtractFonts` . . . 39
6.1.17 `ExtractImages` . . . 39
6.1.18 `FlattenSignatureFields` . . . 39
6.1.19 `ForceCompressionTypes` . . . 40

6.1.20 ForceRecompression ... 40
6.1.21 GetPdf ... 40
6.1.22 ImageStratConserv ... 41
6.1.23 ImageQuality ... 41
6.1.24 IndexedCompressions ... 41
6.1.25 LicenseIsValid ... 42
6.1.26 Linearize ... 42
6.1.27 LinearizeFile ... 42
6.1.28 ListFonts ... 43
6.1.29 ListImages ... 44
6.1.30 MergeEmbeddedFonts ... 45
6.1.31 MonochromeCompression ... 45
6.1.32 MonochromeResolutionDPI ... 45
6.1.33 MonochromeThresholdDPI ... 45
6.1.34 MrcLayerCompression ... 45
6.1.35 MrcLayerQuality ... 46
6.1.36 MrcLayerResolutionDPI ... 46
6.1.37 MrcMaskCompression ... 46
6.1.38 MrcPictCompression ... 46
6.1.39 MrcRecognizePictures ... 47
6.1.40 Open ... 47
6.1.41 OpenMem ... 47
6.1.42 OptimizeResources ... 48
6.1.43 PageCount ... 48
6.1.44 ProductVersion ... 48
6.1.45 Profile ... 48
6.1.46 ReduceColorComplexity ... 49
6.1.47 RemoveNonSymbolicFonts ... 49
6.1.48 RemoveRedundantObjects ... 49
6.1.49 RemoveStandardFonts ... 50
6.1.50 ResolutionDPI ... 51
6.1.51 SaveAs ... 51
6.1.52 SaveInMemory ... 52
6.1.53 SetCMSEngine ... 52
6.1.54 SetInfoEntry ... 53
6.1.55 SetLicenseKey ... 53
6.1.56 SetVersion ... 53
6.1.57 Strip ... 54
6.1.58 SubsetFonts ... 54
6.1.59 ThresholdDPI ... 55
6.1.60 UnembedFont ... 55
6.2 Enumerations ... 55
6.2.1 TPDFColorConversion ... 55
6.2.2 TPDFComprAttempt ... 56
6.2.3 TPDFCompression ... 57
6.2.4 TPDFErrorCode ... 58
6.2.5 TPDFOptimizationProfile ... 58
6.2.6 TPDFPermission ... 62
6.2.7 TPDFStripType ... 63

7 Licensing, Copyright, and Contact ... 65

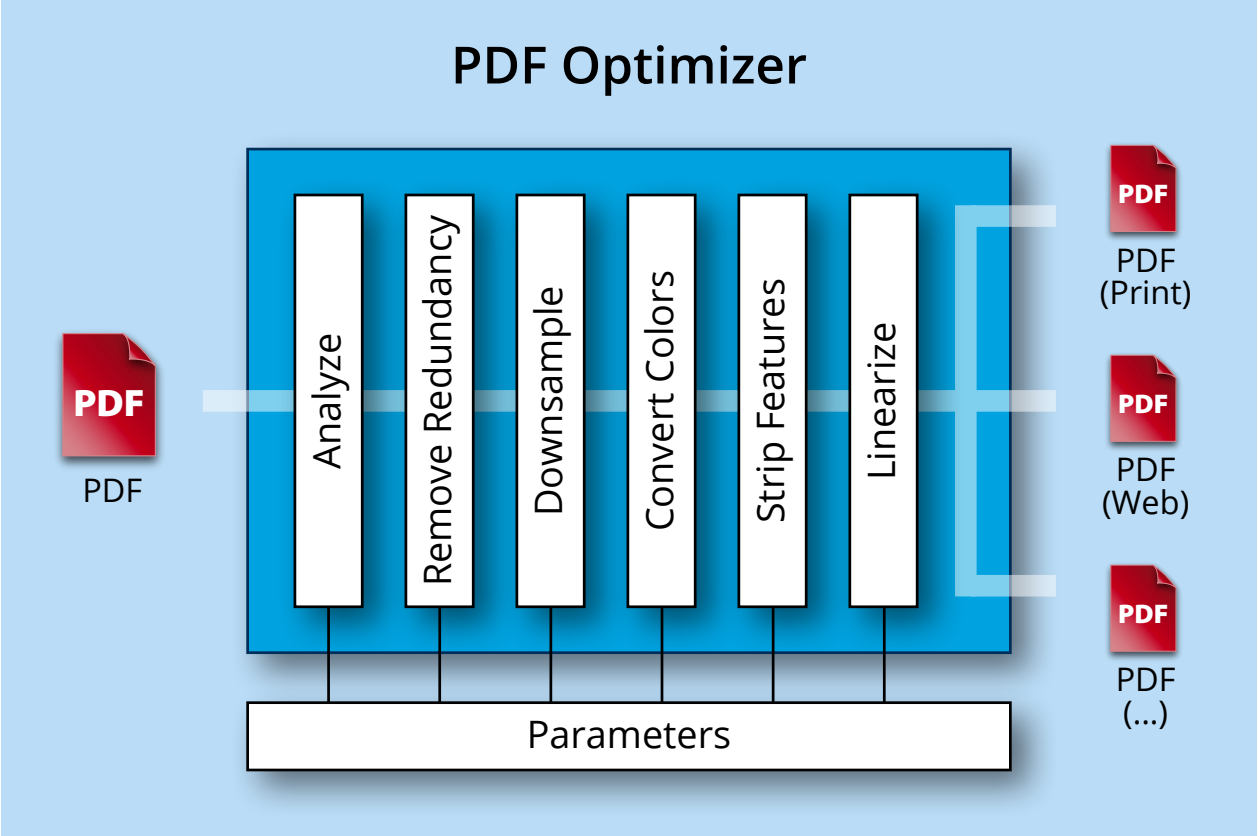
1 Introduction

1.1 Description

The 3-Heights™ PDF Optimizer API optimizes PDF files to enable their use as high resolution files for printing or, with less resolution, for electronic document exchange or space-saving document archiving.

Many processes produce very large PDF files that are not suitable for electronic document exchange. Users are then tempted to convert the PDF documents into other formats, but this only makes the situation even worse. The correct approach, and the easiest, is to optimize large PDF documents.

This process optimizes fonts and images to the best possible size and quality. It also removes redundant document content and "linearizes" PDF documents to enable fast web display.



1.2 Functions

The use of the latest compression algorithms enables the tool to reduce the memory space requirements for images or lessen their resolution, remove redundant and alternative information, optimize fonts through summarization or subsetting, convert colors and linearize the PDF.

1.2.1 Features

- Optimization for Electronic Document Exchange, Web Publishing and Archiving
- Customized compression of bi-tonal, monochrome and color images
- Define image resolution in dots per inch
- Define threshold value for down-sampling

- Set the quality index of lossy compression
- Automatically select best compression type for images
- Perform mixed raster content (MRC) optimization for images
- Remove invisible parts of images
- Linearization (fast web display)
- Compile and subset fonts
- Read encrypted input files
- Encrypt and set access authorization for the output file
- Process memory-resident files
- Removal of:
  - Redundant objects
  - Obsolete objects stemming from previous changes to the file
  - Embedded standard fonts (e.g. Courier, Arial, Times)
  - Embedded, non-symbolic fonts
  - Unnecessary file information
  - Article threads
  - Alternative images
  - Metadata
  - Page piece information
  - Document structure tree including markup
  - Miniature page preview images
  - Spider (web capture) information
- Remove or clear form fields and annotations

Optimize for Printing

- Color conversion (to RGB, CMYK or grayscale)
- Allow high print quality
- Set minimum PDF version of the output file

List and extract parameters

- Fonts and their properties
- Images and their properties
- Error Code
- Number of pages

1.2.2 Formats

Input Formats

- PDF 1.x (e.g. PDF 1.4, PDF 1.5.)

Target Formats

- PDF 1.x (e.g. PDF 1.4, PDF 1.5.)

1.2.3 Compliance

Standards ISO 32000 (PDF 1.7)

1.3 Interfaces

The following interfaces are available

- C
- Java
- .NET
- COM

1.4 Operating Systems

The 3-Heights™ PDF Optimizer API is available for the following operating systems:

- Windows Vista, 7, 8, 8.1, 10 – 32 and 64 bit
- Windows Server 2008, 2008 R2, 2012, 2012 R2, 2016 – 32 and 64 bit
- HP-UX 11 and later PA-RISC2.0 – 32 bit
- HP-UX 11i and later ia64 (Itanium) – 64 bit
- IBM AIX 6.1 and later – 64 bit
- Linux 2.6 – 32 and 64 bit
- Oracle Solaris 2.8 and later, SPARC and Intel
- FreeBSD 4.7 and later (32 bit) or FreeBSD 9.3 and later (64 bit, on request)
- macOS 10.4 and later – 32 and 64 bit

2 Installation and Deployment

2.1 Windows

The 3-Heights™ PDF Optimizer API comes as a ZIP archive containing various files including runtime binary executable code, files required for the developer, documentation and license terms.

1. Download the ZIP archive of the product from your download account at https://www.pdf-tools.com.
2. Unzip the file using a tool like WinZip available from WinZip Computing, Inc. at http://www.winzip.com to a directory on your hard disk where your program files reside (e.g. `C:\Program Files\PDF Tools AG`)
3. Check the appropriate option to preserve file paths (folder names). The unzip process now creates the following subdirectories:

Subdirectory	Description
`bin`	Contains the runtime executable binary code.
`doc`	Contains documentation files.
`include`	Contains header files to in include in your C/C++ project.
`jar`	Contains java archive files for java components.
`lib`	Contains the object file library to include in your C/C++ project.
`samples`	Contains sample programs in various programming languages

   There is the option to download the software as MSI file, which makes the installation easier. Only one version (32 or 64 bit) can be installed using the MSI file. In order to install both versions the ZIP file should be used.
4. Optionally register your license key using the License Management.
5. Identify which interface you are using. Perform the specific installation steps for that interface described in chapter Interface Specific Installation Steps
6. Make sure your platform meets the requirements regarding color spaces described in chapter Color Profiles.

2.2 Unix

This section describes installation steps required on all Unix platforms, which includes Linux, macOS, Oracle Solaris, IBM AIX, HP-UX, FreeBSD and others.

The Unix version of the 3-Heights™ PDF Optimizer API provides two interfaces:

- Java interface
- Native C interface

Here is an overview of the shared libraries and other files that come with the 3-Heights™ PDF Optimizer API:

File Description

Name	Description
bin/libPdfOptimizeAPI.so	This is the shared library that contains the main functionality. The file extension varies depending on the UNIX platform.
doc/*.*	Documentation
jar/POLA.jar	Java API archive.
include/*.h	Contains files to include in your C/C++ Project.

Example code written in different programming languages are available at product page of the PDF Tools AG website.

2.2.1 All Unix Platforms

1. Unpack the archive in an installation directory, e.g. /opt/pdf-tools.com/
2. Copy or link the shared object into one of the standard library directories, e.g:

```
ln -s /opt/pdf-tools.com/bin/libPdfOptimizeAPI.so /usr/lib
```

3. Verify that the GNU shared libraries required by the product are available on your system:
   - *On Linux*:

     ```
     ldd libPdfOptimizeAPI.so
     ```

   - *On AIX*:

     ```
     dump -H libPdfOptimizeAPI.so
     ```

   In case you have not installed the GNU shared libraries yet, proceed as follows:
   a. Go to http://www.pdf-tools.com and navigate to "Support" →"Resources".
   b. Download the GNU shared libraries for your platform.
   c. Extract the archive and copy or link the libraries into your library directory, e.g /usr/lib or /usr/lib64.
   d. Verify that the GNU shared libraries required by the product are available on your system now.
4. Optionally register your license key using the Command Line License Manager Tool.
5. Identify which interface you are using. Perform the specific installation steps for that interface described in chapter Interface Specific Installation Steps
6. Make sure your platform meets the requirements regarding color spaces described in chapter Color Profiles.

2.2.2 macOS

The shared library must have the extension .jnilib for use with Java. We suggest that you create a file link for this purpose by using the following command:

```
ln libPdfOptimizeAPI.dylib libPdfOptimizeAPI.jnilib
```

2.3 Interfaces

The 3-Heights™ PDF Optimizer 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 examples with which programming languages they 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 This interface is available in the Windows version only.
`Java`	The Java interface is available on all platforms.
`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++. This interface is available on all platforms.

2.3.1 Development

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

Req. This file is required for this interface.

Opt. This file is optional. See also table File Description to identify which files are required for your application.

Doc. This file is for documentation only.

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

Files for Development

Name	.NET	Java	COM	C
`bin\PdfOptimizeAPI.dll`	Req.	Req.	Req.	Req.

Files for Development

bin*NET.dll	Req.			
bin*NET.xml	Doc.			
doc*.pdf	Doc.	Doc.	Doc.	Doc.
doc\PdfOptimizeAPI.idl			Doc.	
doc\javadoc*.*		Doc.		
include\pdfoptimizeapi_c.h				Req.
include*.*				Opt.
jar\POLA.jar		Req.		
lib\PdfOptimizeAPI.lib				Req.
samples*.*	Doc.	Doc.	Doc.	Doc.

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

File Description

Name	Description
bin\PdfOptimizeAPI.dll	This is the DLL that contains the main functionality (required).
bin*NET.dll	The .NET assemblies are required when using the .NET interface. The files bin*NET.xml contain the corresponding XML documentation for MS Visual Studio.
doc*.*	Various documentations.
include*.*	Contains files to include in your C / C++ project.
lib\PdfOptimizeAPI.lib	The object file library needs to be linked to the C/C++ project.
jar\POLA.jar	The Java API archive.
samples*.*	Contains sample programs in different programming languages.

[1] These files must reside in the same directory as PdfOptimizeAPI.dll.

2.3.2 Deployment

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

Files for Deployment

Name	.NET	Java	COM	C
bin\PdfOptimizeAPI.dll	Req.	Req.	Req.	Req.
bin*NET.dll	Req.			
jar\POLA.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 into an installation routine such as an MSI file or simple batch script.
4. Perform any interface-specific actions (e.g. registering when using the COM interface).

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 `application.exe`. Color profiles are not used.
2. The application uses the COM interface and is distributed on Windows only.
   - The main DLL `PdfOptimizeAPI.dll` must be distributed.
3. All files are copied to the target location using a batch script. This script contains the following commands:

```
copy application.exe %targetlocation%\.
copy PdfOptimizeAPI.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%\PdfOptimizeAPI.dll.
```

2.4 Interface Specific Installation Steps

2.4.1 COM Interface

Registration Before you can use the 3-Heights™ PDF Optimizer 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 the registration of PdfOptimizeAPI.dll. Note that in Windows Vista and later, the command needs to be executed from an administrator shell.

```
regsvr32 "C:\Program Files\PDF Tools AG\bin\PdfOptimizeAPI.dll"
```

If you are using a 64 bit operating system and would like to register the 32 bit version of the 3-Heights™ PDF Optimizer API, you need to use the `regsvr32` from the directory `%SystemRoot%\SysWOW64` instead of `%SystemRoot%\System32`.[2]

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 `PdfOptimizeAPI.dll`.

2.4.2 Java Interface

The 3-Heights™ PDF Optimizer API requires Java version 1.4 or higher.

For compilation and execution When using the Java interface, the Java wrapper `jar\POLA.jar` needs to be on the CLASSPATH. This can be done by either adding it to the environment variable CLASSPATH, or by specifying it using the switch `-classpath`:

```
javac -classpath ".;C:\pdf-tools\jar\POLA.jar" sample.java
```

For execution Additionally the library `PdfOptimizeAPI.dll` needs be in one of the system's library directories[3] or added to the Java system property `java.library.path`. This can be achieved 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.

```
java -classpath ".;C:\pdf-tools\jar\POLA.jar" ^
 -Djava.library.path=C:\pdf-tools\bin sample
```

2.4.3 .NET Interface

The 3-Heights™ PDF Optimizer API does not provide a pure .NET solution. Instead, it consists of .NET assemblies, which are added to the project and a native DLL, which is called by the .NET assemblies. This has to be accounted for when installing and deploying the tool.

The .NET assemblies (`*NET.dll`) are to be added as references to the project. They are required at compilation time.

`PdfOptimizeAPI.dll` is not a .NET assembly, but a native DLL. It is not to be added as a reference in the project.

The native DLL `PdfOptimizeAPI.dll` is called by the .NET assembly `PdfOptimizeAPINET.dll`.

`PdfOptimizeAPI.dll` must be found at execution time by the Windows operating system. The common way to do this is adding `PdfOptimizeAPI.dll` as an existing item to the project and set its property "Copy to output directory" to "Copy if newer".

Alternatively the directory where `PdfOptimizeAPI.dll` resides can be added to the environment variable `%Path%` or it can simply be copied manually to the output directory.

---

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

[3] On Windows defined by the environment variable PATH and e.g. on Linux defined by LD_LIBRARY_PATH.

2.4.4 C Interface

- The header file `pdfoptimizeapi_c.h` needs to be included in the C/C++ program.
- The library `PdfOptimizeAPI.lib` needs to be linked to the project.
- The dynamic link library `PdfOptimizeAPI.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 used the MSI for the installation, go to Start → 3-Heights™ PDF Optimizer API. . . → Uninstall . . .

If you used the ZIP file: In order to uninstall the product undo all the steps done during installation, e.g. un-register using `regsvr32 -u`, delete all files, etc.

Installing a new version does not require to previously uninstall the old version. The files of the old version can directly be overwritten with the new version. If using the COM interface, the new DLL must be registered, un-registering the old version is not required.

2.6 Color Profiles

The color conversion feature of the 3-Heights™ PDF Optimizer API uses color profiles by default.

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 Optimizer 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 Optimizer API.

2.6.1 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%\spool\drivers\color`
2. directory `Icc`, which must be a direct sub-directory of where the `PdfOptimizeAPI.dll` resides.

Linux and other Unixes

1. `$PDF_ICC_PATH` if the environment variable is defined
2. the current working directory

2.6.2 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:

- http://www.pdf-tools.com/public/downloads/resources/colorprofiles.zip
- http://www.color.org/srgbprofiles.html
- https://www.adobe.com/support/downloads/iccprofiles/iccprofiles_win.html

2.7 Note about the Evaluation License

With the evaluation license the 3-Heights™ PDF Optimizer API automatically adds a watermark to the output files.

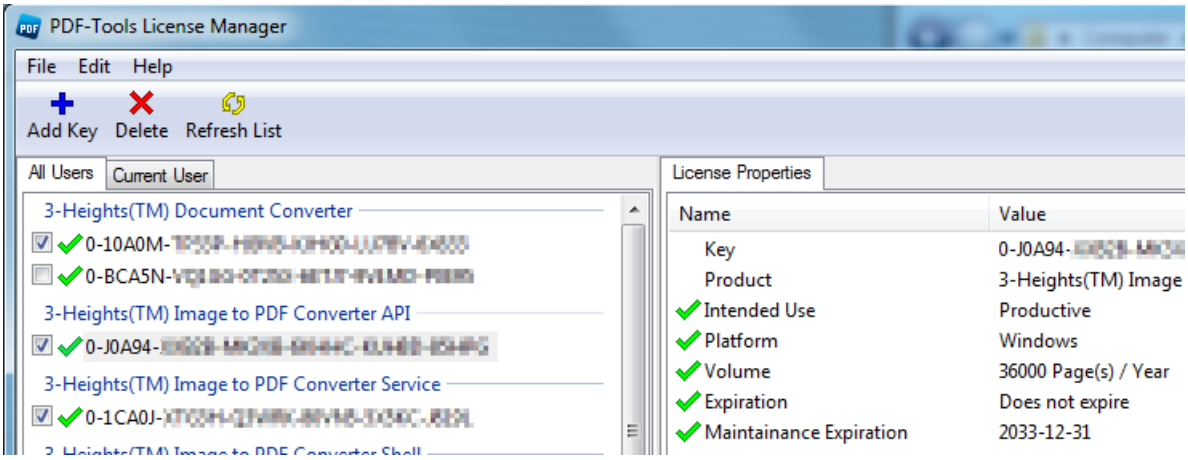
3 License Management

There are three possibilities to pass the license key to the application:

1. The license key is installed using the GUI tool (graphical user interface). This is the easiest way if the licenses are managed manually. It is only available on Windows.
2. The license key is installed using the shell tool. This is the preferred solution for all non-Windows systems and for automated license management.
3. The license key is passed to the application at run-time via the `SetLicenseKey` method. This is the preferred solution for OEM scenarios.

3.1 Graphical License Manager Tool

The GUI tool `LicenseManager.exe` is located in the `bin` directory of the product kit.



3.1.1 List all installed license keys

The license manager always shows a list of all installed license keys in the left pane of the window. This includes licenses of other PDF Tools products. The user can choose between:

- Licenses available for all users. Administrator rights are needed for modifications.
- Licenses available for the current user only.

3.1.2 Add and delete license keys

License keys can be added or deleted with the “Add Key” and “Delete” buttons in the toolbar.

- The “Add key” button installs the license key into the currently selected list.
- The “Delete” button deletes the currently selected license keys.

3.1.3 Display the properties of a license

If a license is selected in the license list, its properties are displayed in the right pane of the window.

3.1.4 Select between different license keys for a single product

More than one license key can be installed for a specific product. The check-box on the left side in the license list marks the currently active license key.

3.2 Command Line License Manager Tool

The command line license manager tool `licmgr` is available in the `bin` directory for all platforms except Windows.

A complete description of all commands and options can be obtained by running the program without parameters:

```
licmgr
```

List all installed license keys:

```
licmgr list
```

The currently active license for a specific product is marked with a star '*' on the left side.

Add and delete license keys:

Install new license key:

```
licmgr store X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

Delete old license key:

```
licmgr delete X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

Both commands have the optional argument `-s` that defines the scope of the action:

g For all users

u Current user

Select between different license keys for a single product:

```
licmgr select X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

3.3 License Key Storage

Depending on the platform the license management system uses different stores for the license keys.

3.3.1 Windows

The license keys are stored in the registry:

- "`HKLM\Software\PDF Tools AG`" (for all users)
- "`HKCU\Software\PDF Tools AG`" (for the current user)

3.3.2 macOS

The license keys are stored in the file system:

- `/Library/Application Support/PDF Tools AG` (for all users)

- `~/Library/Application Support/PDF Tools AG` (for the current user)

3.3.3 Unix/Linux

The license keys are stored in the file system:

- `/etc/opt/pdf-tools` (for all users)
- `~/.pdf-tools` (for the current user)

> **Note:** The user, group and permissions of those directories are set solely by the license manager tool. It may be necessary to change permissions to make the licenses readable for all users. Example:
>
> ```
> chmod -R go+rx /etc/opt/pdf-tools
> ```

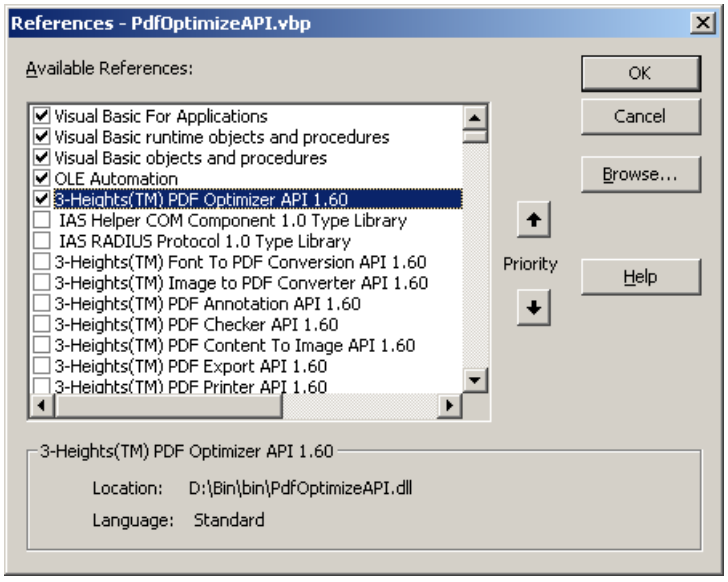
4 Programming Interfaces

4.1 Visual Basic 6

After installing the 3-Heights™ PDF Optimizer API and registering the COM interface (see chapter Installation and Deployment), you find a Visual Basic 6 example `PdfOptimizeAPI.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, here is a quick start guide for you:

1. First create a new Standard-Exe Visual Basic 6 project. Then include the 3-Heights™ PDF Optimizer API component to your project.



2. Draw a new Command Button and optionally rename it if you like.
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.

Example: Simple Visual Basic Sample

```
Private Sub Command1_Click()
    Dim Opt As New PDFOPTIMIZEAPILib.PdfOptimize
    ' Open and analyze the input file.
    Opt.Open "C:\pdf\input.pdf"
    ' Optimize output
    Opt.ColorConversion = eConvRGB
    Opt.BitonalCompressions = eComprAttemptGroup4
    Opt.ContinuousCompressions = eComprAttemptJPEG + eComprAttemptJPEG2000
    Opt.ImageQuality = 75
    Opt.ResolutionDPI = 150
    Opt.ThresholdDPI = 225
    Opt.Linearize = True
    Opt.RemoveRedundantObjects = True
    Opt.SaveAs "C:\out.pdf", "owner", ePermPrint + ePermFillForms
    ' Terminate
```

```
    Opt.Close
End Sub
```

4.2 ASP - VBScript

```
<%@ Language=VBScript %>
<%
  option explicit

  dim pdfOpt
  set pdfOpt = Server.CreateObject("PDFOPTIMIZEAPI.PDFOptimizer")

  if not pdfOpt.Open("http://www.pdf-tools.com /public/downloads/manuals/
pola.pdf", "") then
      Response.Write "<p>"
      Response.Write "Could not open input file." & "<br>"

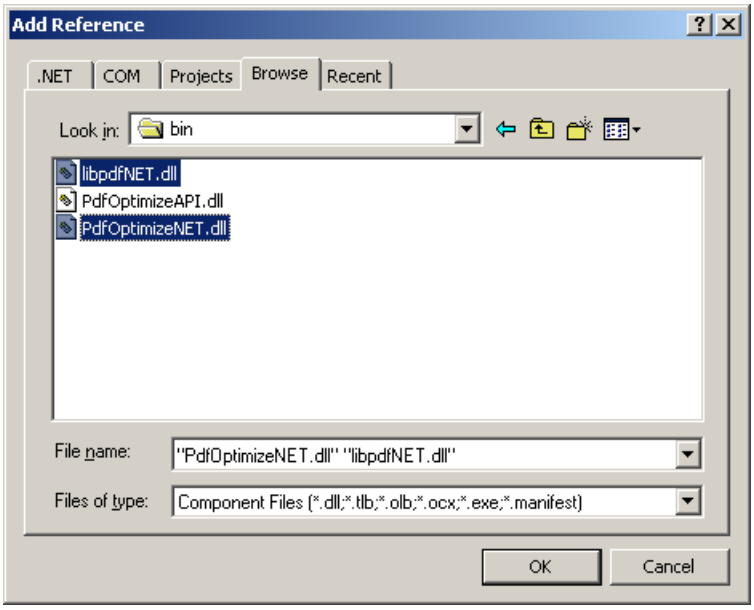
  else
  pdfOpt.RemoveRedundantObjects = True
  pdfOpt.Linearize = True
  if not pdfOpt.SaveAs("C:\temp\optimized.pdf", vbNullString, vbNullString,
   -1)
  then
     Response.Write "<p>"
     Response.Write "Could not save optimized file." & "<br>"
  else
     Response.Write "<p>"
     Response.Write "Optimized output file created <br>"
     Response.Write "</p>"
  end if
end if
%>
```

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 Optimizer API. The easiest for a quick start is to refer to this sample.

In order to create a new project from scratch, do the following steps:

1. Start Visual Studio and create a new C# or VB project.
2. Add references to the .NET assemblies.
   To do so, in the "Solution Explorer" right-click your project and select "Add Reference...". The "Add Reference" dialog will appear. In the tab "Browse", browse for the .NET assemblies `libpdfNET.dll`, `RendererNET.dll`, and `PdfOptimizeAPINET.dll`.
   Add them to the project as shown below:

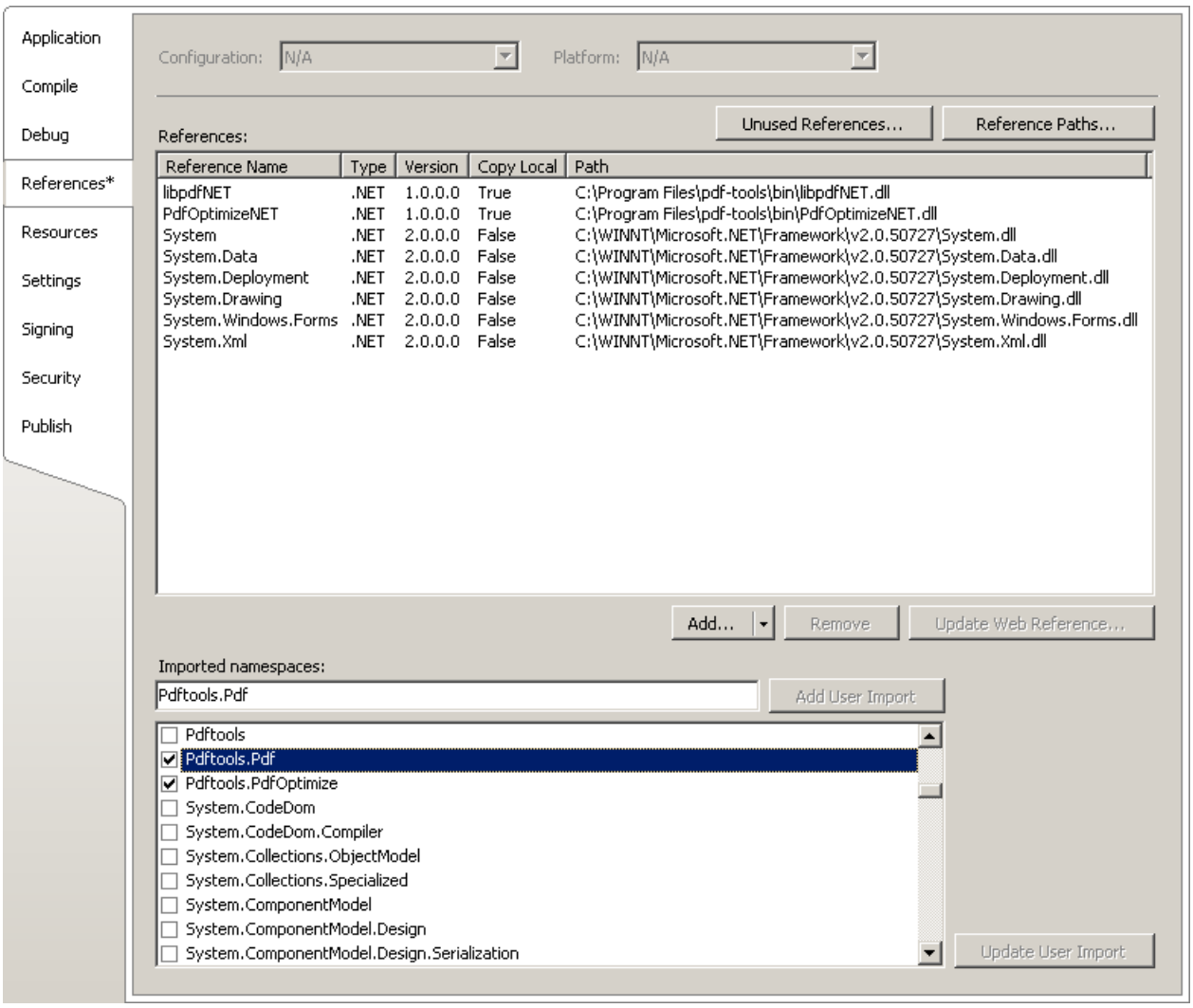


3. Import namespaces (Note: This step is optional, but useful.)
4. Write your code.

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.PdfOptimize`.
   You should now have settings similar as in the screenshot below:



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

Example:

```
Dim opt As New Pdftools.PdfOptimize.Optimizer
opt.Open(...)
...
opt.Close()
```

4.3.2 C#

3. Add the following namespaces:

Example:

```
using Pdftools.Pdf;
using Pdftools.PdfRenderer;
using Pdftools.PdfOptimize;
```

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

Example:

```
using (Optimizer opt = new Optimizer())
{
    opt.Open(...);
    ...
    opt.Close();
}
```

4.3.3 Deployment

This is a guideline on how to distribute a .NET project that uses the 3-Heights™ PDF Optimizer API.

Detailed description

1. The project must be compiled using Microsoft Visual Studio. Hereby it is cruciual that the solution platform (x86 or x64) matches the platform of the native DLL `PdfOptimizeAPI.dll`.
2. The executable is created in the directory `bin\Release`.
3. For deployment, the executable and all .NET assemblies must be copied into the same folder on the target computer. The .NET assemblies of the 3-Heights™ PDF Optimizer API have the file name `*NET.dll`.
4. At runtime, the native DLL `PdfOptimizeAPI.dll` must be found on the target computer by the DLL search sequence. To ensure this, the DLL must either be copied to the folder containing the executable or to a directory on the environment variable `Path` (e.g. `%SystemRoot%\system32`).
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.

Simplified description

1. Compile the project using Microsoft Visual Studio.
2. Deploy your executable and all files in the directory `bin` of the 3-Heights™ PDF Optimizer API's installation directory to the same directory on the target computer.

4.3.4 Troubleshooting: TypeInitializationException

The most common issue when using the .NET interface is that the correct native DLL `PdfOptimizeAPI.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, distinguishable by the inner exception (property `InnerException`):

`System.DllNotFoundException` Unable to load DLL `PdfOptimizeAPI.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 the respective issue.

Troubleshooting: DllNotFoundException

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

Resolve this by either:

- adding `PdfOptimizeAPI.dll` as an existing item to your project and set its property "Copy to output directory" to "Copy if newer", or
- adding the directory where `PdfOptimizeAPI.dll` resides to the environment variable `%Path%`, or
- copying `PdfOptimizeAPI.dll` to the output directory of your project.

Troubleshooting: BadImageFormatException

The exception means, that the native DLL `PdfOptimizeAPI.dll` has the wrong "bitness" (i.e. platform 32 vs. 64 bit). There are two versions of `PdfOptimizeAPI.dll` available: one is 32-bit (name of product kit download contains `WIN32`) and the other 64-bit (name of download contains `X64`). It is crucial, that the platform of the native DLL matches the platform of the application's process.

The platform of the application's process is defined by the project's platform configuration for which there are 3 possibilities:

AnyCPU This means, that the application will run as a 32-bit process on 32-bit Windows and as 64-bit process on 64-bit Windows. When using AnyCPU one has to use a different native DLL, depending on the platform of Windows. This can be ensured 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).

x86 This means, that the application will always run as 32-bit process, regardless of the platform of the Windows installation. The 32-bit DLL runs on all systems, which makes this the simplest configuration. Hence, if an application needs to be portable and does not require any specific 64-bit features, it is recommended to use this setting.

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

5 User's Manual

5.1 Overview of the API

The 3-Heights™ PDF Optimizer API requires a PDF document as input. In this manual, that document is referred to as input document. This input document is processed (optimized) and the result is stored into an output PDF document. (When only Extracting Resources then no output PDF document need be produced.)

Basically, the API is used as follows:

1. Create a `PDFOptimizer` object.
2. Configure optimization settings by setting various properties of this object.
3. Use the object (potentially several times) to process a PDF document.
4. Destroy the object.

Details about how to configure the optimizer is described in How to Optimize PDF Documents and subsequent sections as well as in the Reference Manual.

The optimization process consists of 3 main steps.

1. Open document The document is opened from a file (`Open`) or from memory (`OpenMem`). Decryption of encrypted documents is done automatically. For password-protected encryption, a user password must be provided.

2. Analyze document The document is analyzed. This is done automatically.

3. Optimize and save as new document The optimizer goes through the whole input document and constructs an output document. While doing so, the optimizer transforms objects of the input document according to the optimizer's settings. Often not all the settings are relevant. E.g. if an input document contains color images only, the settings for monochrome and bi-tonal images are not used.

The output document is constructed either as a file (`SaveAs`) or as a block of memory (`SaveInMemory`). The output file name must be different from the input file name. The input document, be it a file or a block of memory, is never changed.

5.2 How to Optimize PDF Documents

5.2.1 Identify Target Application Area

PDF documents are used in a wide variety of application areas, all having different requirements. As a very first step, one should precisely identify the targeted application area. A few typical fields of application are described briefly below. However, PDF documents can also be used in other ways or in combinations of the ones listed below.

Web

All documents related to the web should be kept small in file size. As a consequence they take less storage on the web-server and can be transferred quicker, resulting in shorter download times.

In order to reduce the file size as much as possible, all information that is not required for displaying the document without a visual loss can be removed. This may include:

- Down-sampling images (`ThresholdDPI`, `ResolutionDPI`)
- Clipping images to their visible parts (`ClipImages`)

- Applying compressions algorithms with high compression ratios (BitonalCompressions, ContinuousCompressions, IndexedCompressions)
- Collapsing redundant objects (RemoveRedundantObjects)
- Removing unused resources (OptimizeResources)
- Removing irrelevant information such as article threads, metadata, alternate images, document structure information, etc. (Strip)
- Merging and sub-setting embedded font programs (MergeEmbeddedFonts and SubsetFonts)
- Depending on the PDF documents to be optimized, font programs of embedded standard fonts can even be removed (RemoveStandardFonts).

Additionally, PDF documents can be linearized (Linearize). This is a method of preparing a PDF file in way that pages can be accessed randomly via a PDF viewer web-browser plug-in, i.e. selected pages can be displayed before the whole file is downloaded. For this to work, the PDF viewer web-browser plug-in has to support correct interpretation of linearized PDF.

Documents which are intended to be displayed on a Computer display should be saved in ab RGB (red green blue) color space. RGB is the native form for any light-emitting device, such as computer monitor or television. An RGB image uses three channels and therefore takes up less space than a CMYK (cyan magenta yellow black) image which uses four channels. (ColorConversion)

Printing

For printing applications the file size is not the highest priority. More important is to have a document which prints in a predictable way. This means that correct fonts should be used, colors should look as expected, images should be high in resolution, etc.

For that reason no data from the original document that is used for a well-defined re-production should be removed or altered. Fonts should not be un-embedded, images should not be down-sampled. (Of course there are always exceptions).

For many printing applications it may be advantageous to convert images to the CMYK color space because this is primarily used in systems that reflect light (such as printed paper). (ColorConversion)

In certain documents, the same font is embedded multiple times. If, e.g., a PDF-producing software embeds the same font for each created page, then large multi-page documents contain many copies of a font program. Also, a document can contain a complete font program, of which only very few glyphs are used for display. In such situations, merging and sub-setting font programs can lead to faster printing. (MergeEmbeddedFonts and SubsetFonts)

There are still further ways to decrease the file size:

- Clipping images to their visible parts (ClipImages)
- Compressing uncompressed images, e.g. with a lossless compression type (BitonalCompressions, ContinuousCompressions, IndexedCompressions, see also Supported Image Compression Types)
- Collapsing redundant objects (RemoveRedundantObjects)
- Removing unused resources (OptimizeResources)
- Removing irrelevant information for printing, such as thumbnails, article threads, document structure information, etc. (Strip)

Archiving

Archiving can have varying requisites, such as: Minimize the file size, maximize the reproducibility of the document, minimize the access time to find a specific archived document, etc.

The most common way for archiving a PDF is the PDF/A format, which is defined in the ISO Standard 19005. PDF/A requires fonts to be embedded, metadata to be included and prohibits certain features, like LZW or JPEG2000 compression or alternate images. The 3-Heights™ PDF Optimizer API does not create PDF/A compliant output but can be used, e.g., to reduce the file size prior to converting to PDF/A.

Scanned Documents

For certain types of scanned documents, MRC (mixed raster content) optimization can have a significant impact on file size while still preserve the visual appearance of the document. The 3-Heights™ PDF Optimizer API supports MRC, see also Mixed Raster Content (MRC) Optimization for Images.

Special Requirements

As an example for a specific requirement, the 3-Heights™ PDF Optimizer API supports the restriction of compression types for images in a document to a given list of types. (See `ForceCompressionTypes`, `BitonalCompressions`, `ContinuousCompressions`, and `IndexedCompressions`.)

Another requirement may be to encrypt the resulting PDF and protect it by a user password and/or by an owner password. The 3-Heights™ PDF Optimizer API provides both in the method `SaveAs`.

PDF documents which mainly consist of scanned images to which an OCR (optical character recognition) layer is to be applied at a later time should be optimized in a way that the OCR process of the optimized document works as well as with the original. That means that image compression should either be lossless, or at least perceptually lossless. Perceptually lossless refers to a compression which is lossy, but its visual quality is high enough that neither the human eye nor an OCR engine can distinguish between original and optimized document. (See also Supported Image Compression Types.)

5.2.2 Using Optimization Profiles

The 3-Heights™ PDF Optimizer API provides the notion of "optimization profiles" to quickly arrive at a configuration suitable for many application areas. Once the application area is defined, the optimization profile that best matches the requirements can be identified. See `TPDFOptimizationProfile` for all available profiles and their configuration values. The configuration is done by setting this profile (`Profile`) followed by adjusting individual settings that differ from the profile.

Example: Adapt the "web" profile to not down-sampling bi-tonal images.

```
Dim Opt As New PDFOPTIMIZEAPILib.PDFOptimizer
Opt.OptimizationProfile =
  PDFOPTIMIZEAPILib.TPDFOptimizationProfile.eOptimizationProfileWeb
Opt.BitonalThresholdDPI = -1
'Do some optimization e.g. by calling Opt.Open(...) and Opt.SaveAs(...)
```

5.3 Optimizing Images

For the 3-Heights™ PDF Optimizer API the target compression type of an image is specified by means of the properties `BitonalCompressions`, `ContinuousCompressions`, and `IndexedCompressions`. Values of `TPDFComprAttempt` must be used to set these properties. Several values can be combined by means of a bitwise or operation.

5.3.1 Supported Image Compression Types

In PDF, up to 8 different ways of compressing binary data are supported. (See also PDF Reference 1.7, Chapter 3.3 for more information on these types.)

No Compression (Raw)

Raw means no compression is applied.

DCT (JPEG)

Developer	Joint Photographic Experts Group committee
Version	PDF 1.2, PDF/A-1
Color depth	8, 24 bits per pixel
Compression type	Lossy
Compression algorithm	The image is broken up into blocks that are 8 by 8 samples. On each of these blocks and color channel a discrete cosine transformation (DCT) is applied and its coefficients are quantized. The visual quality of the resulting image depends on the loss of information defined by the step size of the quantization and on the image that is being compressed. The compression can be controlled via an image quality parameter—a value from 1 to 100 (default 75). Typical compression ratios are 15:1 (no perceptible loss of information) to 30:1.
Application area	Sampled continuous-tone pictures (photographs)

Flate (ZIP)

Developer	Flate compression is based on the public-domain zlib / deflate compression method.
Version	PDF 1.2, PDF/A-1
Color depth	1-8, 24 bits per pixel
Compression type	Lossless
Compression algorithm	A lossless data compression algorithm that uses a combination of the LZ77 algorithm and Huffman coding.
Application area	Images

LZW

Developer	Abraham Lempel, Jacob Ziv and Terry Welch Copyright based issues, which expired in most countries in 2003/2004, reduced the popularity of this compression. As one of its consequences it is not included in PDF/A standard.
Version	PDF 1.2
Color depth	2-8 bits per pixel
Compression type	Lossless
Compression algorithm	An indexed based compression that is also used in the GIF and TIFF image formats.
Application area	Gray-scale images, artificial images

CCITT Fax Group 3 and 4

Developer	International Telecommunications Union (ITU), formerly known as the Comité Consultatif International Téléphonique et Télégraphique
Version	PDF 1.0, PDF/A-1
Color depth	1 bit per pixel
Compression type	Lossless
Compression algorithm	**Group3** 1-dimensional version of the CCITT Group 3 Huffman encoding algorithm. **Group 3 2D** 2-dimensional version of the CCITT Group 3 Huffman encoding algorithm. **Group 4** An advanced version of a bi-tonal algorithm based on the CCITT Fax Group 3 2D compression.
Application area	Line-art image, bi-tonal, faxes

JBIG2

Developer	Joint Bi-Level Image Experts Group
Version	PDF 1.4, PDF/A-1
Color depth	1 bit per pixel

Compression type	Lossless
Compression algorithm	The image is broken down into individual symbols, which are stored in a table. A symbol is added to the table if it does not exist yet. If a matching symbol already exists, it is used as a reference. This algorithm works especially well for images with a lot of similar symbols such as scanned text or images that use patterns. Generally JBIG2 provides a better compression ratio than CCITT Group 3 or Group 4 compression. Typical compression ratios for text pages are 20:1 to 50:1.
Application area	Line-art image, bi-tonal

JPEG2000

Developer	Joint Photographic Experts Group committee
Version	PDF 1.5, PDF/A-2
Color depth	8, 24 bits per pixel
Compression type	Lossless if the image quality index is set to `100`. Lossy otherwise
Compression algorithm	JPEG2000 is a wavelet-based image compression standard. It was developed with the intention of superseding the original discrete cosine transform-based JPEG standard.
Application area	Sampled continuous-tone pictures (photographs)

5.3.2 Relevant Factors for the File Size

The size of an image is basically determined by four factors:

The pixel mass The total amount of pixels the image has. An image with a size of 600 by 800 pixels has 480'000 pixels total.

The color depth How many bits are required to describe 1 pixel? The table below gives the answer for different types of images. For example, an RGB image with 600 by 800 pixels requires therefore 600 x 800 x 3 bytes = 1.44 Mbytes in uncompressed format.

Color Space	Description	Bits/Pixel
Bi-tonal	Black and white	1
Indexed	Colors are stored in an index table which usually holds 2 to 256 entries, e.g. GIF.	2-8

Grayscale	Monochrome	8
Color RGB	Color using Red, Green, Blue	24
Color CMYK	Color using Cyan, Magenta, Yellow, Key (=black)	32

The compression type A compression algorithm can compress data (such as an image) to reduce its file size. Such an algorithm belongs to either of the following two classes:

Lossless The original image can be restored exactly.

Lossy The compression modifies the pixels. The original image cannot be restored from the compressed version. This is typically applied to photographic images where the human eye cannot distinguish whether the image was modified. The most common lossy compression is JPEG. The benefit of lossy compression is the higher compression ratio.

See also Supported Image Compression Types.

The content of the image The simpler the image, the better it compresses. For most compression algorithms a simple image (e.g. completely white) compresses much better than a complex image (e.g. a photo).

Examples:

CCITT Fax compression was designed to compress black text written on a white background. The algorithm was optimized under the assumption that a page contains more white pixels than black pixels. Therefore a bi-tonal image with a lot of black does generally not compress as well as in image with more white even if they have the same pixel mass.

JBIG2 compression searches for patterns, and uses them multiple times. For example in a scanned text document the same few dozen of characters are used over and over again. The algorithm is optimized to save frequent patterns more efficiently than rare ones.

5.3.3 Provided Features for Optimizing Images

The 3-Heights™ PDF Optimizer API offers the following possibilities to optimize images:

The pixel mass can be reduced. (It cannot be increased.) This is done by clipping (cropping) the image size to its visible extent and/or by reducing the image resolution.

The resolution defines how many pixels there are in given length of the image. The most common unit for resolution is DPI (dots per inch). If an image has a resolution of 200 DPI, it means when displayed at 100% zoom, there are 200 pixels for 1 inch of image. The higher the resolution, the "sharper" is the image. A monitor has usually a resolution of at least 96 DPI, a laser printer of at least 600 DPI. When the file size matters, a common resolution for color and grayscale images in PDF is 150 DPI (usually higher for bi-tonal).

The process of changing the amount of pixels an image has, is called re-sampling, or down-sampling when the result has less pixels than the original image.

In the 3-Heights™ PDF Optimizer API down-sampling is applied by setting a target resolution and a threshold resolution. The default values are 150 DPI for the target resolution and 225 DPI for the threshold resolution. This means every image that has a resolution of 225 DPI or higher is potentially down-sampled to 150 DPI. Of course, the threshold resolution can be set equal to the target resolution. However there are many cases where down-sampling by just a little bit has disadvantages. In particular, lossy images (e.g. JPEG compression) loose visual quality every time they are newly compressed. On top of that the compressed output can be larger than the input because artifacts introduced by the previous compression(s) are now considered as part of the image

which needs to be compressed and lead to a worse compression even when the resolution is reduced. Per default, the 3-Heights™ PDF Optimizer API will, however, prevent such unnecessary re-sampling.

The color depth can be modified for color images. The color depth can be left unchanged, set to Grayscale (8 bit), RGB (24 bit) or CMYK (32 bit). It cannot be changed to black and white (1 bit).

> **Note:** In certain circumstances, the color depth of the image is not converted, e.g. if the resulting file size increases or if the image is pre-blended with a matte color.

The color complexity can be reduced. By "color complexity" we mean the following hierarchy of possible image pixel contents:

1. All pixels have the same color.
2. All pixels are either black or white.
3. All pixels are colored gray.
4. Pixels have differing colors.

By color complexity reduction we mean that images are converted to their lowest possible color complexity. E.g., a color image with only black and white pixels is converted to a bi-tonal image. Furthermore, an image with color complexity 1 (single color) is down-sampled to one pixel.

Color complexity reduction is also applied to masks and soft masks: Soft masks of complexity 2 (bi-tonal) are converted to masks. Masks and soft masks with complexity 1 (single color) whose color is such that the (soft) mask is opaque are removed.

> **Note:** Currently, color complexity reduction is only carried out for images that have a device color space (DeviceRGB, DeviceGray, or DeviceCMYK) or an indexed color space whose base color space is a device color space.

The compression can be setup independently for the following three image compression types:

Type	Description
Bi-tonal	Black and white images.
Indexed	Images with an indexed (also known as "paletted") color space.
Continuous	Color (RGB and CMYK) images and grayscale images.

Bi-tonal images usually contain text or black and white graphics, indexed images usually contain color graphics such as logos, while continuous images usually contain photographs.

For each of the above image types, several compression algorithms can be set. The 3-Heights™ PDF Optimizer API tries all the given compression algorithms and takes the one that yields the smallest file size. Note that the more compression algorithms are set, the longer the process of optimizing images will take.

Furthermore, a more conservative image processing strategy can be enabled. This strategy prevents all the compression trials if the image has neither been clipped nor down-sampled nor undergone a color-conversion. Hence, if the image has not been altered, then the original image from the input document is taken.

The content of the image cannot be changed directly. However changing the resolution or applying a lossy compression algorithm modifies the content of the image.

Note: Unless forcing of re-compression is enabled, the 3-Heights™ PDF Optimizer API never increases the file size of an image because it chooses the smallest among all tried compression algorithms and the original image in the input file. This means the 3-Heights™ PDF Optimizer API cannot be used to "un-compress" embedded images.

5.3.4 Mixed Raster Content (MRC) Optimization for Images

Some raster images—typically scanned documents—consist mainly of text, possibly in several colors and interspersed with some pictures. Such images are difficult to compress with one single compression type because of the diverse or even conflicting features of different parts of the image.

MRC optimization is a way of breaking such images down into parts, such that each part is well suited for one type of a compression algorithm. With this approach, the resulting file size often can be reduced without significantly reducing the visual quality of the document.

Note: There exists an optimization profile for MRC optimization. See `TPDFOptimizationProfile`.

Note:

- MRC optimization can only be enabled for continuous images, i.e. not for bitonal images and images with an indexed color space.
- MRC optimization may yield unexpected results, e.g. because the input image is not suitable for MRC. As another example, images in the original PDF may be stored as small slices, and MRC optimization fails because the 3-Heights™ PDF Optimizer API has no option to concatenate such image slices.
- A PDF that contains MRC-optimized images is not suited for optical character recognition (OCR) and image extraction.

In the 3-Heights™ PDF Optimizer API, MRC optimization works in three phases as explained below.

Phase 1: Cutting out Pictures

In this phase, the input image is analyzed and rectangular areas containing photographic features are detected. Each detected region is cut out and placed as a separate image in the resulting PDF.

Depending on the input image it is possible that this phase decides that the whole input image consists of one photographic region covering the whole image. In this case, the second phase (Phase 2: Separation into Layers) is omitted.

On the other hand, it is possible, that actual photographic regions present in the input image are not recognized correctly. This can happen for example if a photographic region contains parts with uniform color.

For the cut-out images, a compression type can be set.

Note: The resulting cut pictures are neither down-sampled nor color-converted.

This first phase is optional and can be switched off using the property `MrcRecognizePictures`.

Phase 2: Separation into Layers

For this second phase the image is not supposed to contain any photographic features. Instead, the image is assumed to consist of text and graphic, potentially with varying color.

Now, the whole image is separated into two layers, a foreground and a background layer. Additionally, a mask is created, which can be thought of as a bi-tonal image that is not displayed directly but tells for each pixel whether to show the foreground layer or the background layer.

Example:

Let the image consist of a yellow background with black paragraph text and a title text in red. Then the resulting background layer contains the yellow color only. The foreground layer contains the black text color where the paragraph text is located and the red text color where the title is located. In the mask, pixels for which the foreground layer should be displayed are set to 1, the others are set to 0. I.e. the mask contains 1's where the black and the red text is and 0's everywhere else.

In the resulting PDF the foreground layer, the background layer and the mask are stored as three images and thus are allowed to have different resolution and different compression types. Since all the detailed features have been moved to the mask, it makes sense to down-sample the foreground and background layers and use a low image quality. The mask on the other hand is usually stored with a lossless compression type optimized for text.

Phase 3: Reconstruction

In this phase the results of phase 1 (the cut-out images) and phase 2 (the layers and the mask) are used to synthesize the desired result. If in phase 1, a single photographic region covering the entire image is detected, then the original image is used and the reconstruction is finished. Otherwise, the reconstruction first places the background layer, followed by the foreground layer with the mask. Finally if any cut-images are found they are placed at their respective locations on top of the foreground layer.

5.4 Optimizing Fonts

Every text in a PDF document is written with a font. This font can either be embedded or not embedded in the resources of the PDF. Embedded means a font program is embedded that describes how glyphs are drawn. If a font is not embedded the application rendering the PDF (e.g. 3-Heights™ PDF Viewer or Adobe Acrobat) have to select a replacement font. Therefore the visual appearance of text written with an embedded font is determinable, whereas it is not when the font is not embedded.

A font program can be quite large. An embedded font which contains all WinAnsi characters has a size of about 20-100 Kbytes, if it contains a large Unicode range (e.g. Asian Characters) it can be several Mbytes, whereas an non embedded font requires much less.

This leads to the following ways to optimize fonts:

Remove the embedded font: Removing embedded fonts can reduce the file size of a document, particularly when the document contains many fonts. Removing fonts is best applied to (PDF-) standard fonts, such as Arial, Courier, Courier New, Helvetica, Times, Times New Roman. Removing fonts should not be applied to barcode fonts or fancy types.

> **Note:** PDF/A requires fonts to be embedded.

Subset fonts: Only keep the information in the font program that is required to render the characters that are actually used in text in this document. All unused characters are removed.

Merge fonts: A document can have the same font, or a subset of it, embedded multiple times. This commonly occurs when multiple input document, are merged into one large output document. The 3-Heights™ PDF Optimizer API Tool can merge these fonts into one font (if they can be merged).

5.5 Extracting Resources

The 3-Heights™ PDF Optimizer API can extract resources, such as images or fonts. This is achieved using the corresponding calls `ExtractImages` and `ExtractFonts`.

These resources are extracted unaltered from the PDF document. In particular this means:

- Fonts are not converted to installable fonts, i.e. extracted fonts cannot be installed and used on the operating system. (That would in most cases also be a legal issue.)
- Images are extracted from the resources, without the context of the page. This means they do not inherit the resolution of the image on the page in the PDF document. (Note that the same image could be used multiple times in the document at different resolutions anyway). Also images on the PDF page could possibly be clipped (i.e. not the complete image is visible), or stretched or rotated, etc. All these PDF operators affecting the visual appearance of the image on the page are neglected.

Resources are extracted to the current directory. How to set the current directory depends on the programming language and the OS.

Example: In C# such a command is

```
System.IO.Directory.SetCurrentDirectory("C:\\temp\\");
```

6 Reference Manual

Note: This manual describes the COM interface only. Other interfaces (C, Java, .NET) however 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 PDFOptimizer Interface

6.1.1 BitonalCompression

[Deprecated] Property (get, set): `TPDFCompression BitonalCompression`

Deprecated in Version 4.6, use `BitonalCompressions`.

6.1.2 BitonalCompressions

Property (get, set): `TPDFCompressionAttempt BitonalCompressions`
Default: `eComprNone`

Get or set the compression types for bi-tonal images. See also enumeration `TPDFComprAttempt`.

Several values can be combined with bitwise or operators. The following values are allowed:

- `eComprAttemptNone`
- `eComprAttemptRaw`
- `eComprAttemptFlate`
- `eComprAttemptLZW`
- `eComprAttemptGroup3`
- `eComprAttemptGroup4`
- `eComprAttemptSource`
- `eComprAttemptJBIG2`

Other values are ignored.

Note: This property is affected when setting a `Profile`.

During optimization, all set compression types are tried and the one resulting in the least memory footprint is taken.

Typically, CCITT Group 4 or JBIG2 is used for bi-tonal compression. Due to the simpler algorithm CCITT Group 4 has the advantage of being faster. JBIG2 can achieve compression ratios that are up to twice as high as CCITT Group 4 at the cost of longer computation time.

6.1.3 BitonalResolutionDPI

Property (get, set): `Float BitonalResolutionDPI`
Default: `200`

Get or set the target resolution in dots per inch (DPI) after re-sampling images for bi-tonal images. See also `ResolutionDPI`.

Note: This property is affected when setting a `Profile`.

6.1.4 BitonalThresholdDPI

Property (get, set): `Float BitonalThresholdDPI`
Default: `-1`

Get or set the threshold resolution in dots per inch (DPI) to selectively activate re-sampling for bi-tonal images. The value `-1` deactivates re-sampling for bi-tonal images. See also `ThresholdDPI`.

Note: This property is affected when setting a `Profile`.

6.1.5 ClipImages

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

Get or set the option to clip images. When enabled, then invisible parts of images are clipped (cropped). While this does not affect visual parts of images, it may have a minor visual impact because clipped images are re-compressed. Pre-blended images are not clipped.

Enabling this property will also enable the `OptimizeResources` property.

Note: This property is affected when setting a `Profile`.

6.1.6 Close

Method: `Boolean Close()`

Close an opened input file. If the document is already closed the method does nothing.

Returns:

`True` The file was closed successfully.

`False` Otherwise.

6.1.7 ColorCompression

[Deprecated] Property (get, set): `TPDFCompression ColorCompression`

Deprecated in Version 4.6, use `ContinuousCompressions`.

6.1.8 ColorConversion

Property (get, set): `TPDFColorConversion ColorConversion`
Default: `eConvNone`

Get or set the color conversion. Color conversion is applied to images. Image can be not converted (`eConvNone`), converted to RGB (`eConvRGB`), to CMYK or gray scale. Color key masked images are not color converted. Pre-blended images can be converted from RGB to Grayscale, if the force conversion feature is set.

Note: This property is affected when setting a `Profile`.

Color conversion is mostly used for specific application areas. E.g. in the printing industry the CMYK color space is used, since it represents the colors that printer devices commonly support. If colors are exclusively used for the monitor, the RGB color space should be used.

See also enumeration `TPDFColorConversion`.

6.1.9 ColorResolutionDPI

Property (get, set): `Float ColorResolutionDPI`
Default: `150`

Get or set the target resolution in dots per inch (DPI) after re-sampling images for color images. See also `ResolutionDPI`.

Note: This property is affected when setting a `Profile`.

6.1.10 ColorThresholdDPI

Property (get, set): `Float ColorThresholdDPI`
Default: `-1`

Get or set the threshold resolution in dots per inch (DPI) to selectively activate re-sampling for color images. The value `-1` deactivates re-sampling for color images. See also `ThresholdDPI`.

Note: This property is affected when setting a `Profile`.

6.1.11 ContinuousCompressions

Property (get, set): `TPDFComprAttempt ContinuousCompressions`
Default: `eComprAttemptNone`

Get or set the compression types to be tried for continuous images, i.e. RGB, CMYK, and grayscale images. See also `TPDFComprAttempt`.

Several values can be combined with bitwise or operators. The following values are allowed:

- `eComprAttemptNone`
- `eComprAttemptRaw`
- `eComprAttemptJPEG`
- `eComprAttemptFlate`
- `eComprAttemptJPEG2000`
- `eComprAttemptSource`
- `eComprAttemptMRC`

Other values are ignored.

During optimization, all set compression types are tried and the one resulting in the least memory footprint is taken.

Note: This property is affected when setting a `Profile`.

6.1.12 CompressionQuality

[Deprecated] Property (get, set): `Single CompressionQuality`

Deprecated, use `ImageQuality` instead.

6.1.13 ConvertToCFF

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

Convert embedded Type1 (PostScript) fonts to Type1C (Compact Font Format). This reduces the file size.

Note: This property is affected when setting a `Profile`.

6.1.14 DitheringMode

Property (get, set): `TPDFDitheringMode DitheringMode`
Default: `eDitherNone`

This option enables or disables dithering when down-sampling bi-tonal images.

The only values supported are `eDitherNone` and `eDitherFloydSteinberg`.

Some bi-tonal images try to evoke the impression of different levels of gray by randomly setting pixels to black. If dithering is applied during down-sampling then the gray levels of such images are preserved better. If dithering is switched off then lines (e.g. text glyphs) are preserved better.

Note: This property is affected when setting a Profile.

6.1.15 ErrorCode

Property (get): `TPDFErrorCode ErrorCode`

This property can be accessed to receive the latest error code. 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 Optimizer API.

6.1.16 ExtractFonts

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

Get or set whether to extract embedded fonts. Depending on the font type, the extracted font has one of the following three formats: `fnt‹objno›.ttf` or `fnt‹objno›.pfb` or `fnt‹objno›.cff`, where `‹objno›` is the number of the PDF object of the font.

6.1.17 ExtractImages

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

Get or set whether to extract images. Depending on the compression, the extracted image has one of the following formats: `img‹objno›.tif` or `img‹objno›.jpg`, where `objno` is the number of the PDF object of the image.

6.1.18 FlattenSignatureFields

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

A signature in a PDF consist of two parts:

a. The invisible digital signature in the PDF.
b. The visual appearance that was attributed to the signature.

Part (a) can be used by a viewing application, to verify that a document has not changed since it has been signed and report this to the user. Part (b) is merely a "decorative" element on the page without further significance.

When optimizing a PDF, the PDF is altered and hence the digital signature is broken. Therefore, the 3-Heights™ PDF Optimizer API removes all signatures, including parts (a) and (b).

When the property `FlattenSignatureFields` is set to `True`, then digital signatures (parts (a)) are still removed, but their visual appearances (parts (b)) are flattened. I.e. the latter are retained and drawn as non-editable graphic onto the page.

> **Note:** The resulting PDF can be misleading as it visually appears to be signed, but it has no digital signature and hence, a viewer application does not report any broken signature. In most cases, such a behavior is undesirable.

6.1.19 ForceCompressionTypes

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

If this option is set, then re-compression of images is forced if an image in the input PDF has a compression type that differs from the compression types given in `ContinuousCompressions`, `BitonalCompressions`, or `IndexedCompressions`. Use this option if you want to allow only the given compression types for images in the output PDF.

> **Note:** This property is affected when setting a `Profile`.

6.1.20 ForceRecompression

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

If set, all images are always recompressed. If not set (default), images are only recompressed if the resulting image is smaller than the original, i.e. requires less bytes to store in the file.

> **Note:** This property is affected when setting a `Profile`.

6.1.21 GetPdf

Method: `Variant GetPdf()`

Get the output file from memory. See also method SaveInMemory.

Returns:

A byte array containing the output PDF. In certain programming languages, such as Visual Basic 6, the type of the byte array must explicitly be Variant.

6.1.22 ImageStratConserv

[Deprecated] Property (get, set): Boolean ImageStratConserv
Default: False

Deprecated in Version 4.7.

Enables or disables a more conservative strategy for processing images. When enabled then the compression types set in BitonalCompressions, ContinuousCompressions, and IndexedCompressions are only tried if the image has either been clipped (ClipImages), re-sampled (ResolutionDPI, ThresholdDPI), or if it has undergone a color conversion (ColorConversion), otherwise the original input image is taken as is. See also Provided Features for Optimizing Images.

Note: This property is affected when setting a Profile.

6.1.23 ImageQuality

Property (get, set): Single ImageQuality
Default: 75

Get or set the quality index of lossy compression types. This value ranges from 1 to 100 and is applied to JPEG and JPEG2000 compression. For JPEG2000, a quality index of 100 means lossless compression. JPEG compression is always lossy.

Note: This property is affected when setting a Profile.

6.1.24 IndexedCompressions

Property (get, set): TPDFComprAttempt IndexedCompressions
Default: eComprAttemptFlate

Get or set the compression types for images that have an indexed ("palette") color space. See also TPDFComprAttempt.

Several values can be combined with bitwise or operators. The following values are allowed:

- eComprAttemptNone
- eComprAttemptRaw
- eComprAttemptFlate

Note: This property is affected when setting a Profile.

- eComprAttemptLZW
- eComprAttemptSource

Other values are ignored.

During optimization, all set compression types are tried and the one resulting in the least memory footprint is taken.

6.1.25 LicenseIsValid

Property (get): Boolean LicenseIsValid
Static

Check if the license is valid.

6.1.26 Linearize

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

Get or set whether to linearize the PDF output file, i.e. optimize file for fast web access.

A linearized document has a slightly larger file size than a non-linearized file and provides the following main features:

- When a document is opened in a PDF viewer of a web browser, the first page can be viewed without downloading the entire PDF file. In contrast, a non-linearized PDF file must be downloaded completely before the first page can be displayed.
- When another page is requested by the user, that page is displayed as quickly as possible and incrementally as data arrives, without downloading the entire PDF file.

Note: This property is affected when setting a Profile.

6.1.27 LinearizeFile

Method: Boolean LinearizeFile(String FileName, String Password, String OutFileName, String UserPw, String OwnerPw, String PermissionFlags)

Linearize a PDF file and save the result as a new PDF file, which is optimized for fast web view. This is a standalone function and cannot be combined with any other functions or properties.

Parameters:

FileName [String] The input PDF file name, i.e. the name of document that is read.

Password [String] (optional) The user or owner password of the input file name. A password must be provided if the input file is protected by a user password, otherwise an empty string can be passed as argument.

OutFileName [String] The output PDF file name, i.e. the name of the linearized document that is written.

UserPw [String] (optional) The user password of the output PDF file.

OwnerPw [String] (optional) The owner password of the output PDF file.

PermissionFlags [String] (optional) The permission flags if the document is encrypted and secured by an owner password.

Additional information about UserPw, OwnerPw and PermissionFlags can be founding the method SaveAs.

Returns:

True A linearized PDF file was successfully created.

False Otherwise.

6.1.28 ListFonts

Method: Boolean ListFonts(String FileName)

List all fonts included in the document and write them as a list to a text file

Parameter:

FileName [String] The file name of the output text file, to which the information should be stored. The list contains the header line "FontName, FontType, Encoding, IsCID, IsEmbedded, IsSubsetted, Filename".

The meanings of these columns are:

FontName The name of the font, such as Arial-BoldMT or TimesNewRomanPS-BoldMT.

FontType The font type, such as TrueType or Type1

Encoding The encoding of the font, such as WinAnsiEncoding or MacRomanEncoding.

IsCID The font is CID (character identifier) keyed. This value is either CID or Non-CID.

IsEmbedded The font program for this font is embedded in the PDF document. This value is either Embedded or Non-Embedded.

FileName The file name of the font program. This is the name under which the font is saved to file in case the font is extracted and saved. Only embedded fonts can be extracted. The file name consists of the prefix fnt, the object number and the file type which is one of .ttf, .pfb or .cff.

Example: fnt38.ttf

Returns:

True The font information was successfully extracted and written to the output text file.

False Otherwise.

6.1.29 ListImages

Method: `Boolean ListImages(String FileName)`

List all images included in the document and write them as a list to a text file.

Parameter:

`FileName` `[String]` The file name of the output text file, to which the information should be stored. The list contains the header line:

`PageNumber, ObjectNumber, Width, Height, BitsPerComponent, ColorSpace, Resolution, Filter, ImageSize, CompressedSize, CompressionRatio, FileName`

The meanings of these columns are

PageNumber The page number in the PDF on which the image occurs.

ObjectNumber The PDF object number which contains this image.

Width The width of the image in dots (pixels).

Height The height of the image in dots (pixels).

BitsPerComponent The amount of bits that are used per component. This value is for example 1 for bi-tonal images and 8 for gray-scale and color images.

ColorSpace The color space can be one of `DeviceGray`, `DeviceRGB`, `DeviceCMYK`, `ICCBased`, `Indexed`.

Resolution The ratio of amount of pixels divided by the length of the image on the page.

Example: An image is 300 dots (pixel) wide and takes 1 inch (2.54cm) on the page in the PDF. This image has a resolution of 300 DPI (dots per inch). If the same image is stretched to 2 inch, its resolution is 150 DPI.

Filter The compression filter, for example: `FlateDecode`, `DCTDecode`, `CCITTFaxDecode`.

ImageSize The size in bytes of the uncompressed image.

CompressedSize The size in bytes of the compressed image.

CompressionRatio The ratio compressed size divided by uncompressed size.

FileName The file name of the image. This is the name under which the image is saved to file in case the image is extracted and saved. The file name consists of the prefix `img`, followed by the PDF object number, and the extension which is one of `.jpg` or `.tif`, depending on the extracted image type. Example: `img19.jpg`, `img21.tif`.

Returns:

`True` The image information was successfully extracted and written to the output text file.

`False` Otherwise.

6.1.30 MergeEmbeddedFonts

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

Merge embedded font programs. Font programs can be merged, if they originate from the same font, e.g. they are of the same type, have the same name and encoding. Merging of Type1 (PostScript) and TrueType fonts is supported.

Note: This property is affected when setting a Profile.

6.1.31 MonochromeCompression

[Deprecated] Property (get, set): `TPDFCompression MonochromeCompression`

Deprecated in Version 4.6, use ContinuousCompressions.

6.1.32 MonochromeResolutionDPI

Property (get, set): `Float MonochromeResolutionDPI`
Default: `150`

Get or set target resolution in dots per inch (DPI) after re-sampling images for monochrome images. See also ResolutionDPI.

Note: This property is affected when setting a Profile.

6.1.33 MonochromeThresholdDPI

Property (get, set): `Float MonochromeThresholdDPI`
Default: `-1`

Get or set the threshold resolution in dots per inch (DPI) to selectively activate re-sampling for monochrome images. The value `-1` deactivates re-sampling for monochrome images. See also ThresholdDPI.

Note: This property is affected when setting a Profile.

6.1.34 MrcLayerCompression

Property (get, set): `TPDFCompression MrcLayerCompression`
Default: `eComprJPEG2000`

Get or set the compression type for MRC foreground and background layers. See TPDFCompression for possible values. See also Mixed Raster Content (MRC) Optimization for Images.

Note: This property is affected when setting a Profile.

6.1.35 MrcLayerQuality

Property (get, set): Short MrcLayerQuality
Default: 10

Get or set the image quality for MRC foreground and background layers when using a lossy compression type. This is a value between 0 and 100.

See also Supported Image Compression Types, Relevant Factors for the File Size, and Mixed Raster Content (MRC) Optimization for Images.

Note: This property is affected when setting a Profile.

6.1.36 MrcLayerResolutionDPI

Property (get, set): Float MrcLayerResolutionDPI
Default: 70

Get or set the target resolution in DPI for down-sampling MRC foreground and background layers. If set to -1 then no down-sampling is performed. See also Provided Features for Optimizing Images and Mixed Raster Content (MRC) Optimization for Images.

Note: This property is affected when setting a Profile.

6.1.37 MrcMaskCompression

Property (get, set): TPDFCompression MrcMaskCompression
Default: eComprGroup4

Get or set the compression type for MRC masks. See TPDFCompression for possible values. See also Mixed Raster Content (MRC) Optimization for Images.

Note: This property is affected when setting a Profile.

6.1.38 MrcPictCompression

Property (get, set): TPDFCompression MrcPictCompression
Default: eComprJPEG

Get or set the compression type for MRC cut-out pictures. See TPDFCompression for possible values. See also Mixed Raster Content (MRC) Optimization for Images.

Note: This property is affected when setting a Profile.

6.1.39 MrcRecognizePictures

Property (get, set): Boolean MrcRecognizePictures
Default: True

Get or set the option to recognize photographic regions when doing MRC. When set to False then Phase 1: Cutting out Pictures is omitted, i.e., no regions of photographic content are identified. See also Mixed Raster Content (MRC) Optimization for Images.

Note: This property is affected when setting a Profile.

6.1.40 Open

Method: Boolean Open(String Filename, String Password)

Open a PDF file, i.e. make the objects contained in the document accessible. If a 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 property ErrorCode for additional information.

6.1.41 OpenMem

Method: Boolean OpenMem(Variant MemBlock, String Password)

Open a PDF 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.

6.1.42 OptimizeResources

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

Get or set whether resources should be optimized. If set, unused resources such as images, fonts, and color spaces are removed. Also content streams are re-built.

Note: This property is affected when setting a Profile.

6.1.43 PageCount

Property (get): `Long PageCount`

Get the number of total pages of the document. If no document is opened, it returns `0`.

6.1.44 ProductVersion

Property (get): `String ProductVersion`

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

6.1.45 Profile

Property (set): `TPDFOptimizationProfile Profile`
Default: `eOptimizationProfileDefault`

With this property one of the predefined optimization profiles can be set. If a profile is set then all the properties listed in TPDFOptimizationProfile (table Profile Settings) are set to their respective values. Properties not listed in this table are left unchanged.

One way of quickly arriving at a specific setting is to first set the `Profile` and then adapt the configuration by setting some of the individual properties.

Example: In C#.

```
using (Optimizer opt = new Optimizer())
{
    opt.Profile = PDFOptimizationProfile.eOptimizationProfileWeb;
    opt.Linearize = false;
    // ... perform some optimization now
}
```

6.1.46 ReduceColorComplexity

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

This property is used to enable color complexity reduction of images. (See also Provided Features for Optimizing Images.)

If enabled then images with device color spaces (DeviceRGB, DeviceCMYK, or DeviceGray) and indexed images with a device color space as base color space are analyzed and if possible converted as follows:

- An image with DeviceRGB or DeviceCMYK color space in which all pixels are gray is converted to a grayscale image with DeviceGray color space.
- An image that contains only black and white pixels is converted into a bitonal image.
- An image in which all the pixels have the same color is down-sampled to one pixel.

Furthermore, images' masks and soft masks are optimized as follows:

- A soft mask that contains only black and white pixels is converted to a mask.
- A (soft) mask that is opaque is removed.

Note: This property is affected when setting a `Profile`.

6.1.47 RemoveNonSymbolicFonts

[Deprecated] Property (get, set): `Boolean RemoveNonSymbolicFonts`
Default: `False`

This property has no effect.

6.1.48 RemoveRedundantObjects

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

Get or set whether redundant objects should be removed. If this property is set to `True`, duplicate objects are removed in order to reduce the file size.

Note: This property is affected when setting a `Profile`.

6.1.49 RemoveStandardFonts

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

Get or set whether to remove the font programs of all embedded standard fonts, such as Arial, Courier, CourierNew, Helvetica, Symbol, Times, TimesNewRoman and ZapfDingbats. (A complete list is given below.) The fonts are replaced with one of the 14 PDF Standard Fonts, all of which have no associated font program. Un-embedding a font decreases the file size.

Note: This property is affected when setting a `Profile`.

A PDF Viewer must be able to display these 14 PDF Standard Fonts correctly. Therefore enabling this property usually should not visually alter the PDF when it is displayed.

Un-embedding the font works based on the font's Unicode information. I.e. the un-embedded font's characters are mapped to those of the original font with the same Unicode. Therefore, only fonts with Unicode information will be un-embedded by the 3-Heights™ PDF Optimizer API. However, if a font's Unicode information is not correct, un-embedding may lead to visual differences. Whether or not a font's Unicode information is correct can be verified by extracting text that uses the font. Suitable tools for this purpose are for instance the 3-Heights™ PDF Extract Tool or an interactive PDF viewer.

If the extracted text is meaningful, the font's Unicode information is correct and unembedding of the font will not lead to visual differences.

List of Candidate Font Base Names for Removing

- Arial
- Arial,Bold
- Arial,BoldItalic
- Arial,Italic
- Arial-Bold
- Arial-BoldItalic
- Arial-BoldItalicMT
- Arial-BoldMT
- Arial-Italic
- Arial-ItalicMT
- ArialMT
- Courier
- Courier,Bold
- Courier,BoldItalic
- Courier,BoldOblique
- Courier,Italic
- Courier,Oblique
- Courier-Bold
- Courier-BoldOblique
- Courier-Oblique
- CourierNew
- CourierNew,Bold
- CourierNew,BoldItalic
- CourierNew,Italic
- CourierNew-Bold
- CourierNew-BoldItalic
- CourierNew-Italic
- CourierNewPS-BoldItalicNT
- CourierNewPS-BoldMT
- CourierNewPS-ItalicMT
- CourierNewPSMT
- Helvetica
- Helvetica,Bold
- Helvetica,BoldItalic
- Helvetica,BoldOblique
- Helvetica,Italic
- Helvetica,Oblique
- Helvetica-Bold
- Helvetica-BoldItalic
- Helvetica-BoldOblique
- Helvetica-Italic
- Helvetica-Oblique
- Symbol
- SymbolMT
- Times,Bold
- Times,BoldItalic
- Times,Italic
- Times-Bold
- Times-BoldItalic
- Times-Italic
- Times-Roman
- TimesNewRoman
- TimesNewRoman,Bold
- TimesNewRoman,BoldItalic

- TimesNewRoman,Italic
- TimesNewRoman-Bold
- TimesNewRoman-BoldItalic
- TimesNewRoman-Italic
- TimesNewRomanPS
- TimesNewRomanPS-Bold
- TimesNewRomanPS-BoldItalic
- TimesNewRomanPS-BoldItalicMT
- TimesNewRomanPS-BoldMT
- TimesNewRomanPS-Italic
- TimesNewRomanPS-ItalicMT
- TimesNewRomanPSMT
- ZapfDingbats

6.1.50 ResolutionDPI

Property (get, set): `Single ResolutionDPI`
Default: (Different defaults apply to different image types)

Get or set the resolution in DPI (dots per inch) after re-sampling images.

This property affects all three image compression types (`BitonalResolutionDPI`, `ColorResolutionDPI`, `MonochromeResolutionDPI`).

A typical value for the resolution when optimizing for the web is `150` DPI. For printing typically no re-sampling is applied (see property `ThresholdDPI`). Pre-blended images, images with a color key mask, mask, and soft mask images are not re-sampled.

Note: This property is affected when setting a `Profile`.

When getting `ResolutionDPI`, the property returns the target resolution in DPI for color images.

6.1.51 SaveAs

Method: `Boolean SaveAs(String FileName, String UserPw, String OwnerPw, TPDFPermission PermissionFlags)`

Save the currently opened document.

Parameters:

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

`UserPw` `[String]` (optional) Set the user password of the PDF document. If this parameter is omitted, the default password is used. Use `""` to set no password.

`OwnerPw` `[String]` (optional) Set the owner password of the PDF document. If this parameter is omitted, the default password is used. Use `""` to set no password.

`PermissionFlags` `[TPDFPermission]` (optional) The permission flags.

By default no encryption is used (`-1`). The permissions that can be granted are listed at the enumeration `TPDFPermission`. To not encrypt the output document, set PermissionFlags to `ePermNoEncryption`, user and owner password to `""`. In order to allow high quality printing, flags `ePermPrint` and `ePermDigitalPrint` need to be set.

Returns:

`True` The opened document could successfully be saved to file.

False Otherwise. One of the following occurred[4]:

- The output file cannot be created.
- PDF_E_FILECREATE: Failed to create the file.

6.1.52 SaveInMemory

Method: Boolean SaveInMemory()

Save the output PDF in memory. After the Close call it can be accessed using the method GetPdf.

Returns:

True The document could be saved in memory successfully.

False Otherwise.

6.1.53 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, however, 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 section 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
```

[4] This is not a complete list. If SaveAs returns False, it is recommended to abort the processing of the file and log the error code and error message.

```
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.

6.1.54 SetInfoEntry

Method: `Boolean SetInfoEntry(String Key, String Value)`

Set a key-value pair in the document info dictionary. Values of predefined keys are also stored in the XMP metadata.

Popular entries specified in the PDF Reference 1.7 and accepted by most PDF viewers are `"Title"`, `"Author"`, `"Subject"`, `"Creator"` (sometimes referred to as Application), and `"Producer"` (sometimes referred to as PDF Creator).

Parameters:

Key [String] The key as a string.

Value [String] The value as a string.

Example in C#:

```
opt.SetInfoEntry("Producer", "Me, myself, and I");
```

6.1.55 SetLicenseKey

Method: `Boolean SetLicenseKey(String LicenseKey)`

Set the license key.

6.1.56 SetVersion

Method: `Boolean SetVersion(String PDFVersion)`

Set the minimum PDF version of the created PDF output file. Supported values for the string are `"1.1"` to `"1.7"`[5]. There are three parameters that influence the version of the PDF output file:

- The value set using this method
- The PDF version of the input file
- Other optimizer settings (e.g. JBIG2 requires PDF 1.4, JPEG2000 requires PDF 1.5)

The maximum of the three values above sets the PDF version in the output file.

Example: Input PDF is version 1.5 and the following settings are applied

```
SetVersion("1.4")
```

The output file is PDF version 1.5.

Example: Input PDF is version 1.4 or lower and the following settings are applied

```
SetVersion("1.4")
```

The output file is PDF version 1.4.

Example: Input PDF is version 1.3 and the following settings are applied

```
ColorCompression = eComprJPEG2000
SetVersion("1.4")
```

If `input.pdf` contains color images to which JPEG2000 compression is applied, the output file will be version 1.5. Otherwise it will be version 1.4.

6.1.57 Strip

Property (get, set): `TPDFStripType Strip`
Default: `0`

Get or set the stripping mode. This mode can be configured to remove unneeded data of a PDF document such as Threads, Metadata, the PieceInfo, the StructTreeRoot entry, embedded Thumbs and the SpiderInfo entry. Also this mode is used to indicate whether to flatten form fields, links, and other annotations. Multiple values of `TPDFStripType` can be combined with the bitwise or operator.

Note: This property is affected when setting a `Profile`.

6.1.58 SubsetFonts

Property (get, set): `Boolean SubsetFonts`
Default: `false`

[5] PDF 1.4 corresponds to Acrobat version 5, PDF 1.5 to Acrobat version 6, etc.

This property influences two optimizations related to subsetted fonts:

- Subset embedded fonts.
- Merge embedded font programs of different subsets of the same font, granted they can be merged.

Sub-setting refers to removing those glyphs in a font that are not actually used in any text contained in the PDF.

Note: This property is affected when setting a Profile.

6.1.59 ThresholdDPI

Property (get, set): Single ThresholdDPI
Default: -1

Set the threshold in DPI (dots per inch) to selectively activate re-sampling. Only images with a resolution above the threshold DPI will be re-sampled. This property affects all three image compression types (BitonalThresholdDPI, ColorThresholdDPI, MonochromeThresholdDPI). The value -1 deactivates re-sampling.

Note: This property is affected when setting a Profile.

6.1.60 UnembedFont

Method: Boolean UnembedFont(String FontName)

Remove the embedded font program for the font given in FontName.

Warning: The output document may not display correctly on certain systems.

6.2 Enumerations

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

6.2.1 TPDFColorConversion

TPDFColorConversion Table

TPDFColorConversion	Description
Const eConvNone	None
Const eConvRGB	Red Green Blue
Const eConvCMYK	Cyan Magenta Yellow Key
Const eConvGray	Gray

6.2.2 TPDFComprAttempt

In contrast to TPDFCompression, TPDFComprAttempt is meant to be used as a bit-field, i.e. values can be composed with the bitwise or operator (Or in Visual Basic). Use this enumeration to compose values for the BitonalCompressions, ContinuousCompressions, and IndexedCompressions properties.

TPDFComprAttempt Table

TPDFComprAttempt	Description
eComprAttemptNone	Exclude from processing
eComprAttemptRaw	No Compression (Raw)
eComprAttemptJPEG	DCT (JPEG)
eComprAttemptFlate	Flate (ZIP)
eComprAttemptLZW	LZW
eComprAttemptGroup3	CCITT Fax Group 3 (CCITT Fax Group 3 and 4)
eComprAttemptGroup3_2D	CCITT Fax Group 3 2D (CCITT Fax Group 3 and 4)
eComprAttemptGroup4	CCITT Fax Group 4 (CCITT Fax Group 3 and 4)
eComprAttemptJBIG2	JBIG2 (Supported in PDF 1.4 or later)
eComprAttemptJPEG2000	JPEG2000 (Supported in PDF 1.5 or later, not supported in PDF/A-1)

TPDFComprAttempt Table

eComprAttemptMRC	In contrast to the values 0-8, this is not a single compression format. Instead, this enables MRC optimization on color and monochrome images. (See Mixed Raster Content (MRC) Optimization for Images) Application area: Scanned documents.
eComprAttemptSource	In contrast to the values 0-8, this is not a single compression format. Instead, this tells 3-Heights™ PDF Optimizer API to use the same compression as the original input image.

6.2.3 TPDFCompression

Compression types as occurring in PDF.

> **Note:** Not all image formats/color depths support all compression types. See also chapter Supported Image Compression Types.

TPDFColorConversion Table

TPDFColorConversion	Description
eComprRaw	No compression
eComprJPEG	Joint Photographic Expert Group
eComprFlate	Flate compression
eComprLZW	Lempel-Ziv-Welch
eComprGroup3	CCITT Fax Group 3
eComprGroup3_2D	CCITT Fax Group 3 2D
eComprGroup4	CCITT Fax Group 4
eComprJBIG2	Joint Bi-level Image Experts Group
eComprJPEG2000	JPEG2000
eComprUnknown	Unknown compression

6.2.4 TPDFErrorCode

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, e.g. no valid output file was created. 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's 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 Optimizer API, most of which are listed here:

TPDFErrorCode Table

TPDFErrorCode	Description
PDF_S_SUCCESS	The operation was completed successfully.
LIC_E_NOTSET, LIC_E_NOTFOUND, ...	Various license management related errors.
PDF_E_FILEOPEN	Failed to open the file.
PDF_E_FILECREATE	Failed to create the file.
PDF_OPT_E_ANNOTAPPEAR	Cannot create appearance for annotation.
PDF_OPT_W_RMSIGANNOT	A signature annotation was removed.
PDF_W_NOENCRYPTION	The file is PDF/A and must not be encrypted.

6.2.5 TPDFOptimizationProfile

TPDFOptimizationProfile

TPDFOptimizationProfile	Description
eOptimizationProfileDefault	Minimal optimization. This is the default profile.
eOptimizationProfileWeb	Optimization for the Internet
eOptimizationProfilePrint	Optimization for print
eOptimizationProfileMax	Optimization for maximal memory size reduction
eOptimizationProfileMRC	MRC (Mixed Raster Content) optimization of images. See also Mixed Raster Content (MRC) Optimization for Images.

Note: When setting a profile with the property `Profile` then all the properties listed in Profile Settings are set to their respective value. Values not listed are left unchanged.
Also note that gray values in parentheses, although set, have no effect because images are excluded from processing or image down-sampling is disabled due to a threshold set to `-1`.

Profile Settings

	eOptimizationProfileDefault	eOptimizationProfileWeb	eOptimizationProfilePrint	eOptimizationProfileMax	eOptimizationProfileMRC
BitonalCompressions:					
eComprAttemptNone	✓				✓
eComprAttemptGroup4		✓	✓	✓	
eComprAttemptJBIG2		✓		✓	
eComprAttemptSource		✓	✓	✓	
ContinuousCompressions:					
eComprAttemptNone	✓				
eComprAttemptJPEG		✓	✓	✓	
eComprAttemptJPEG2000		✓		✓	
eComprAttemptMRC					✓
eComprAttemptSource		✓	✓	✓	
IndexedCompressions:					
eComprAttemptNone	✓				
eComprAttemptFlate		✓	✓	✓	

Profile Settings

	eOptimizationProfileDefault	eOptimizationProfileWeb	eOptimizationProfilePrint	eOptimizationProfileMax	eOptimizationProfileMRC
eComprAttemptLZW				✓	
eComprAttemptSource		✓	✓	✓	
BitonalResolutionDPI	(200)[6]	200	(200)[6]	160	(200)[6]
BitonalThresholdDPI	-1	280	-1	220	-1
MonochromeResolutionDPI	(150)[6]	150	(150)[6]	130	(150)[6]
MonochromeThresholdDPI	-1	210	-1	180	-1
ColorResolutionDPI	(150)[6]	150	(150)[6]	130	(150)[6]
ColorThresholdDPI	-1	210	-1	180	-1
ImageQuality	75				
ColorConversion	None	RGB	CMYK	None	None
ClipImages	False	True	True	True	True
ReduceColorComplexity	False	True	True	True	False
ForceRecompression	False				
ForceCompressionTypes	False				
DitheringMode	eDitheringNone				
MrcLayerCompression	eComprJPEG2000				
MrcLayerResolutionDPI	70				
MrcMaskCompression	eComprGroup4				

Profile Settings

	eOptimizationProfileDefault	eOptimizationProfileWeb	eOptimizationProfilePrint	eOptimizationProfileMax	eOptimizationProfileMRC
MrcLayerQuality	10				
MrcRecognizePictures	True				
MrcPictCompression	eComprJPEG				
ConvertToCFF	False	True	True	True	False
MergeEmbeddedFonts	False	True	True	True	True
RemoveStandardFonts	False	False	False	True	False
SubsetFonts	False	True	True	True	True
OptimizeResources	False	True	True	True	True
Linearize	False	True	False	False	False
RemoveRedundantObjects	False	True	True	True	True
Strip:					
eStripThreads		✓	✓	✓	✓
eStripMetadata		✓		✓	
eStripPieceInfo		✓	✓	✓	✓
eStripStructTree		✓	✓	✓	✓
eStripThumb		✓	✓	✓	✓
eStripSpider		✓	✓	✓	✓
eStripAlternates		✓		✓	

Profile Settings

	eOptimizationProfileDefault	eOptimizationProfileWeb	eOptimizationProfilePrint	eOptimizationProfileMax	eOptimizationProfileMRC
eStripOutputIntents				✓	
eStripAnnots				✓	
eStripForms				✓	
eStripLinks					

6.2.6 TPDFPermission

An enumeration for permission flags. If a flag is set, the permission is granted.

TPDFPermission Table

TPDFPermissionFlag	Description
ePermNoEncryption	Do not encrypt the document
ePermAll	Grant all permissions
ePermPrint	Low resolution printing
ePermModify	Changing the document
ePermCopy	Content copying or extraction
ePermAnnotate	Annotations
ePermFillForms	Filling of form fields
ePermSupportDisabilities	Support for disabilities

[6] These values, although set, have no effect because down-sampling of images is disabled.

TPDFPermission Table

ePermAssemble	Document assembly
ePermDigitalPrint	High resolution printing

Changing permissions or granting multiple permissions is done using a bitwise or operator. Changing the current permissions in Visual Basic should be done like this:

Allow Printing

```
Permission = Permission Or ePermPrint
```

Prohibit Printing

```
Permission = Permission And Not ePermPrint
```

To disable encryption you should overwrite all flags

```
Permission = ePermNoEncryption
```

6.2.7 TPDFStripType

TPDFStripType Table

TPDFStripType	Description
eStripThreads	Strip article threads
eStripMetadata	Strip meta data
eStripPieceInfo	Strip page piece info (private application data)
eStripStructTree	Strip document structure tree (incl. Mark-up)
eStripThumb	Strip thumbnails
eStripSpider	Strip spider (web capture) info
eStripAlternates	Strip alternate images
eStripForms	Strip and flatten form fields
eStripLinks	Strip and flatten link annotations
eStripAnnots	Strip and flatten annotations except form fields and links

TPDFStripType Table

`eStripFormsAnnots`	Strip and flatten form fields, links, and annotations. This implies `eStripForms`, `eStripLinks`, `eStripAnnots`
`eStripOutputIntents`	Strip the document output intents
`eStripAll`	Strip everything (all of the above)

7 Licensing, Copyright, and Contact

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

PDF Tools AG 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.

Licensing and Copyright

The 3-Heights™ PDF Optimizer API is copyrighted. This user's manual is also copyright protected; it may be copied and given away provided that it remains unchanged including the copyright notice.

Contact

PDF Tools AG
Kasernenstrasse 1
8184 Bachenbülach
Switzerland
http://www.pdf-tools.com
pdfsales@pdf-tools.com