PDF-TOOLS.COM
Premium PDF Technology

3-Heights™ PDF Optimization API

Version 4.5

User Manual

Contact: pdfsupport@pdf-tools.com

Owner: **PDF Tools AG**

Kasernenstrasse 1
8184 Bachenbülach
Switzerland

http://www.pdf-tools.com

1 Table of Content

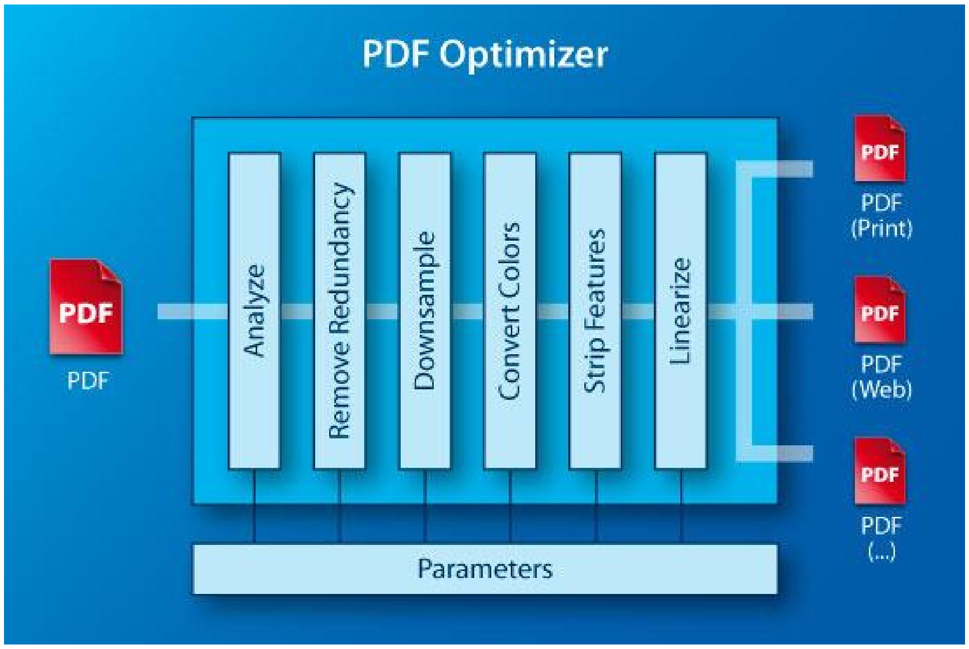
2 Introduction

2.1 Description

The 3-Heights™ PDF Optimization 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.



2.2 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.

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

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)

Compliance

Standards: ISO 32000 (PDF 1.7)

2.3 Interfaces

The following interfaces are available:

- C
- Java
- .NET
- COM

2.4 Operating Systems

- Windows XP, Vista, 7, 8, 8.1 - 32 and 64 bit
- Windows Server 2003, 2008, 2008 R2, 2012, 2012 R2 - 32 and 64 bit
- HP-UX 11 and later PA-RISC2.0 32 bit or HP-UX 11i and later ia64 (Itanium) 64 bit
- IBM AIX 5.1 and later (64 bit)
- Linux (32 and 64 bit)
- Mac OS X 10.4 and later (32 and 64 bit)
- Sun 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)

3 Installation

3.1 Windows

The retail version of the 3-Heights™ PDF Optimization API comes as a ZIP archive containing various files including runtime binary executable code, documentation and license terms.

1. Download the ZIP archive of the product from your download account at www.pdf-tools.com.
2. Open the ZIP archive.
3. Check the appropriate option to preserve file paths (folder names) and unzip the archive to a local folder (e.g. *C:\program files\pdf-tools\*).
4. The unzip process now creates the following subdirectories:

Bin:	Contains the runtime executable binary code
Doc:	Contains documentation files
Include:	Contains files to include in your C / C++ project
Samples:	Contains Visual Basic 6.0 sample program

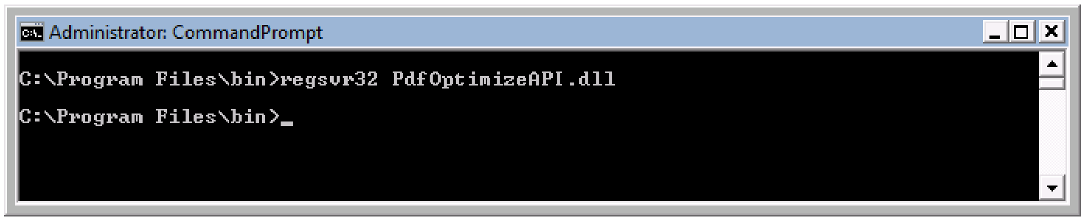
General

Here is an overview of the relevant files that come with the 3-Heights™ PDF Optimization Tool:

bin\PdfOptimizeAPI.dll	This is the DLL that contains the main functionality (required).
Bin\pdcjk.dll	This DLL contains support for Asian languages (optional). It is loaded from the module path.
*Bin*NET.dll*	The .NET assemblies (required if using the .NET interface).
Lib\PDFOptimizeAPI.lib	Import library for C programs.
Jar\POLA.jar	Java API archive.
Doc\PdfOptimizeAPI.idl	COM interface definition.
Include\PdfOptimizeAPI.h	C API include file.
Include\PdfOptimizeAPI_c.h	COM API include file.
Include\PdfOptimizeAPI_i.c	COM API identifier definitions.
Include\pdferror.h	Supplementary C header-file containing error codes.

COM Interface

Before you can use the 3-Heights™ PDF Optimization API component in your COM application program you have to register the component using the regsvr32 program that is provided with the Windows operating system in the directory System32. On Windows Vista and 7, this needs to be done in Administrator mode. The following screenshot shows the registration of the PDF Optimization DLL *PdfOptimizeAPI.dll*.

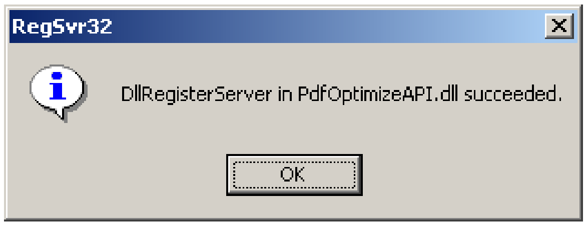


```
C:\Program Files\bin>regsvr32 PdfOptimizeAPI.dll

C:\Program Files\bin>_
```

If you are using a 64 bit Windows platform and would like to register the 32 bit version of the 3-Heights™ PDF Optimization API, it is required to use regsvr32 from WOW64 instead from System32.

If the registration process succeeds the following box is displayed:



DllRegisterServer in PdfOptimizeAPI.dll succeeded.

The installation process is now complete.

Java Interface

When using the Java interface, the Java-wrapper *jar\POLA.jar* needs to be on the CLASSPATH. *PDFOptimizeAPI.dll* needs to be on the environment variable PATH.

.NET Interface

The 3-Heights™ PDF Optimization 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. See also chapter *"Getting Started"*.

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 *PdfOptimizeNET.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.

Native C Interface

The header file *pdfoptimizeapi_c.h* needs to be included in the C program. The library *PdfOptimizeAPI.lib* needs to be linked to the project.

Uninstall, Install a New Version

In order to uninstall the product undo all the steps done during installation, e.g. unregister using regsvr32 –u, delete all files, etc.

Note that an expired evaluation DLL cannot be unregistered. If you would like to unregister an expired evaluation DLL, download a new (non-expired) evaluation version, overwrite the old version and unregister it.

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; unregistering the old version is not required.

3.2 Unix

Here is an overview of the shared libraries and other files that come with the PDF Optimization API:

bin/PdfOptimizeAPI.so	This is the shared library that contains the main functionality.
Jar/POLA.jar	Java API archive.
Include/.h*	C API include file.

All Unix Platforms

1. Unpack the archive in an installation directory, e.g. */usr/pdftools.com/*
2. Copy or link the shared object into one of the standard library directories, e.g:

   ```
   ln -s /usr/pdftools.com/bin/libPdfOptimizeAPI.so /usr/lib
   ```
3. In case you have not yet installed the GNU shared libraries, get a copy of these from http://www.pdf-tools.com; extract the shared images and copy or link them into */usr/lib* or */usr/local/lib*.

Mac OS/X

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
```

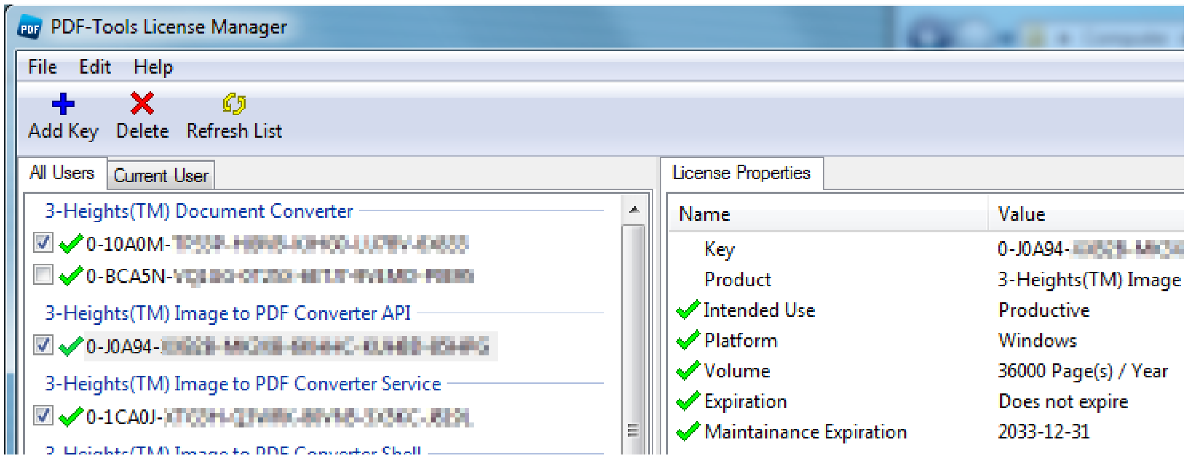
4 License Management

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 runtime via the "LicenseKey" property. This is the preferred solution for OEM scenarios.

4.1 Graphical License Manager Tool

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



3-Heights(TM) Document Converter
- ☑ ✓ 0-10A0M-[redacted]
- ☐ ✓ 0-BCA5N-[redacted]

3-Heights(TM) Image to PDF Converter API
- ☑ ✓ 0-J0A94-[redacted]

3-Heights(TM) Image to PDF Converter Service
- ☑ ✓ 0-1CA0J-[redacted]

3-Heights(TM) Image to PDF Converter Shell

Name	Value
Key	0-J0A94-[redacted]
Product	3-Heights(TM) Image
✓ Intended Use	Productive
✓ Platform	Windows
✓ Volume	36000 Page(s) / Year
✓ Expiration	Does not expire
✓ Maintainance Expiration	2033-12-31

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.

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.

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.

Select between different license keys for a single product

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

4.2 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 ist 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
```

4.3 License Key Storage

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

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)

Mac OS X

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)

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 explicitly 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
```

5 Getting Started and User's Manual

5.1 How to Optimize PDF Documents

Identify Target Application Area

PDF documents are used in a wide variety of application areas, which all have different requirements. As very first step, one should clearly identify what application area documents need be optimized for. 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. Small means they take less storage on the web-server and can be transferred quicker. Shorter download times are appreciated by all users.

  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 includes removing font programs of embedded fonts, down-sampling images and applying compressions algorithms with high compression ratios.

  PDF documents can also be linearized, a method which refers to preparing the 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.

  Documents which are intended to be displayed at the monitor should be saved in the RGB color space. RGB is the native form for any device that emits light, (such as computer monitor or television), it uses three channels and uses therefore less space than CMYK which uses four channels.

- ***Printing:*** In the printing industry the file size is not the highest priority. More important is to have a document which prints in a predictable way. This means the correct fonts should be used, colors should look as expected, 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).

  Colors should be converted to the CMYK color space which is primarily used in systems that reflect light (such as printers).

  There are still ways to lower the file size, e.g. by applying a lossless compression for images or by removing irrelevant information for printing, such as thumbnails, meta data, file attachments, etc.

- ***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 and color profiles to be embedded, metadata to be included and prohibits certain items, like LZW or JPEG2000 compression or alternate images.

The 3-Heights™ PDF Optimization API does not create PDF/A compliant output.

- ***OCR:*** 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.

Apply Adequate Optimization Settings

The following table shows a suggestion what settings should be applied for optimizing in different application areas. These settings are by no means valid for all situations.

	Web	**Printing**	**Archiving**
Color Space	RGB	CMYK	leave as is
Resolution	72 – 150 dpi for color and grey scale images 300 dpi for bi-tonal images	leave as is or set a maximum (e. g. 600 dpi)	leave as is
Fonts	don't embed standard fonts, embed special fonts (e. g. barcodes)	embed	*
Color compression	JPEG or JPX	leave compressed images unchanged, but apply a lossless compression (e.g. flate) to all uncompressed images	
Monochrome compression	JPEG or JPX		
Bi-tonal compression	JBIG2	leave as is **	leave as is **
Quality	For JPEG: 75, for JPX: 30	leave as is	leave as is
Redundant objects	remove	remove	remove
Article thread	strip	strip	*
Form fields	leave as is	strip and flatten	strip and flatten
Alternate images	strip	leave as is	*
Meta data	strip	strip	leave as is
Page piece info	strip	strip	*
Document structure tree	strip	strip	*
Thumbnails	strip	strip	strip
Spider	strip	strip	*

*) Setting depends on the document and type of archive. E.g. if one million similar invoice documents are archived, it might not make much sense to embed the same font one million times.

**) Applying JBIG2 compression to bi-tonal images always yields in a smaller size than CCITT G4. JBIG2 is lossless, but supported only in PDF version 1.4 and newer (Acrobat 5).

5.2 Overview of the API

What Is the 3-Heights™ PDF Optimization API About?

One of the main intents of the 3-Heights™ PDF Optimization API is to reduce the file size of a PDF document. Another is to optimize it for a specific field of application (e.g. Internet, Printing, etc.). For that purpose this API offers various options to optimize embedded resources such as fonts or images.

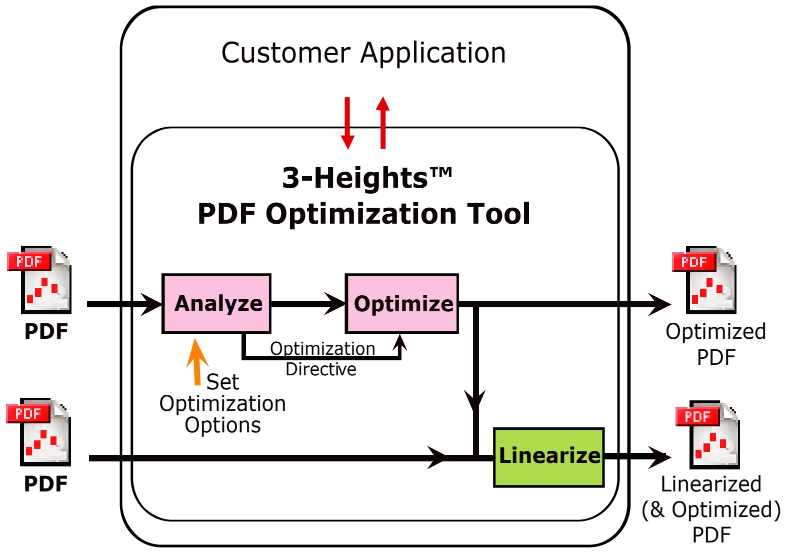
5.3 How Does the API Work?

The 3-Heights™ PDF Optimization API requires a PDF document as input. In this manual, that document is referred to as input-document. In the graphic below that's the document on the left hand side.

The optimization process consists of 3 main steps.

Step 1 Open document

The document is opened from file or from memory. If the document is encrypted, it is decrypted. If the document is encrypted with a user password[1], a password must be passed to the open call.



Step 2 Analyze document

The document is analyzed. This is done automatically.

Step 3 Optimize and save as new document

The Optimization API provides a series of properties, such as compression types and resolutions for different types of images, properties to optimize resources, linearization, etc. You can think of these properties as of a filter. Any object in the input-document is optimized according to these settings. Normally 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, neither any settings related to fonts.

As a result the Optimization API creates a new PDF which is referred to as output-document. This output-document can be a file or a can be kept in memory.

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

The output-document is optimized according to the settings defined in the Optimization API.

The input-document is never changed by the 3-Heights™ PDF Optimization API. The output-document must be different from the input-document.

5.4 Optimizing PDF Files

Relevant Factors for the File Size

The file size of a PDF heavily depends on its content and on how the PDF is constructed internally. Often embedded font programs, embedded color profiles, and images have the highest impact.

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

1. ***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.
2. ***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

3. ***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 chapter *Supported Image Compression Types*.
4. ***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.

Optimizing Raster Images

The 3-Heights™ PDF Optimization Tool offers the following possibilities to optimize raster images:

1. ***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 is, the "sharper" 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 Optimization Tool 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 Optimization Tool will, however, prevent such unnecessary re-sampling.

2. ***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 that 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.

3. ***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 Optimization Tool 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.

4. ***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 Optimization Tool 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 Optimization Tool cannot be used to "un-compress" embedded images.

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: MRC optimization can only be enabled for continuous images, i.e. not for bi-tonal 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 Optimization Tool 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 Optimization Tool, 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.

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.

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:

1. ***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.

2. ***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.

3. ***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™ Optimization Tool can merge these fonts into one font (if they can be merged).

5.5 Extracting Resources

The 3-Heights™ PDF Optimization 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. In C# such a command is:

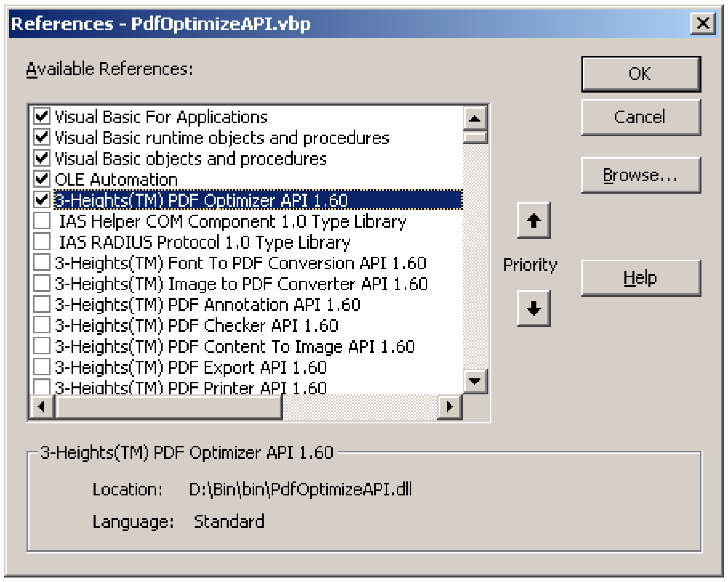
```
System.IO.Directory.SetCurrentDirectory(„C:\\temp\\“);
```

6 Programming Interfaces

6.1 Visual Basic

After installing the 3-Heights™ PDF Optimization API, you find a Visual Basic example *PdfOptimizeAPI.vbp* in the directory “samples”. You can either use this sample as the basis for an application, or you can start from scratch.

If you start from scratch, first create a new Standard-Exe Visual Basic 6 project. Then include the 3-Heighth™ PDF Image Optimization API component to your project.



- [✓] Visual Basic For Applications
- [✓] Visual Basic runtime objects and procedures
- [✓] Visual Basic objects and procedures
- [✓] OLE Automation
- [✓] 3-Heights(TM) PDF Optimizer API 1.60
- [ ] IAS Helper COM Component 1.0 Type Library
- [ ] IAS RADIUS Protocol 1.0 Type Library
- [ ] 3-Heights(TM) Font To PDF Conversion API 1.60
- [ ] 3-Heights(TM) Image to PDF Converter API 1.60
- [ ] 3-Heights(TM) PDF Annotation API 1.60
- [ ] 3-Heights(TM) PDF Checker API 1.60
- [ ] 3-Heights(TM) PDF Content To Image API 1.60
- [ ] 3-Heights(TM) PDF Export API 1.60
- [ ] 3-Heights(TM) PDF Printer API 1.60

3-Heights(TM) PDF Optimizer API 1.60
Location: D:\Bin\bin\PdfOptimizeAPI.dll
Language: Standard

Draw a new Command Button and optionally rename it if you like. Now double-click the Command Button and insert the few lines of code below. All that you need to change is the path of the two file names.

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
```

6.2 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
%>
```

6.3 .NET

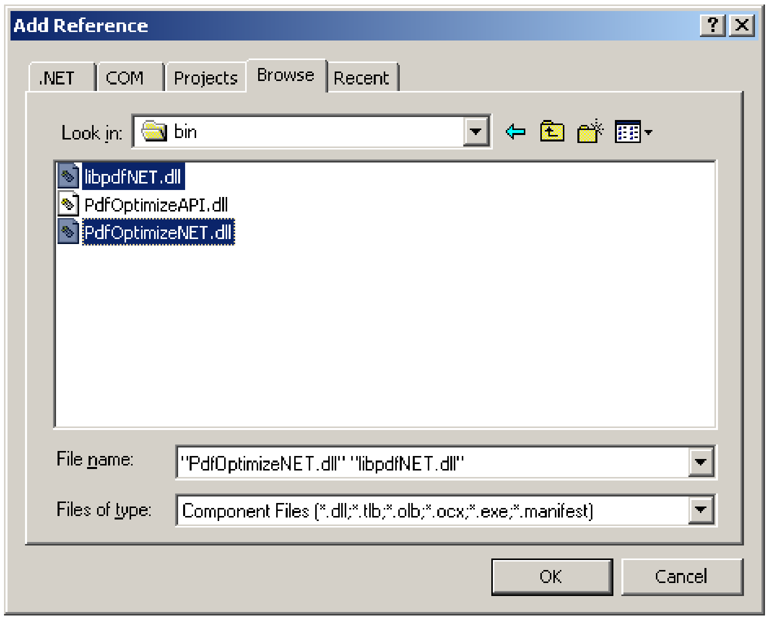
As opposed to previous versions, the Windows build numbers 1.7.1.* and later provide a .NET interface.

There should be at least one .NET sample for MS Visual Studio 2005 available in the ZIP archive of the Windows Version of the 3-Heights™ PDF Optimization 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 a reference 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* and *PdfOptimizeNET.dll* add them to the project as shown below:



libpdfNET.dll
PdfOptimizeAPI.dll
PdfOptimizeNET.dll

File name: "PdfOptimizeNET.dll" "libpdfNET.dll"

Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

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

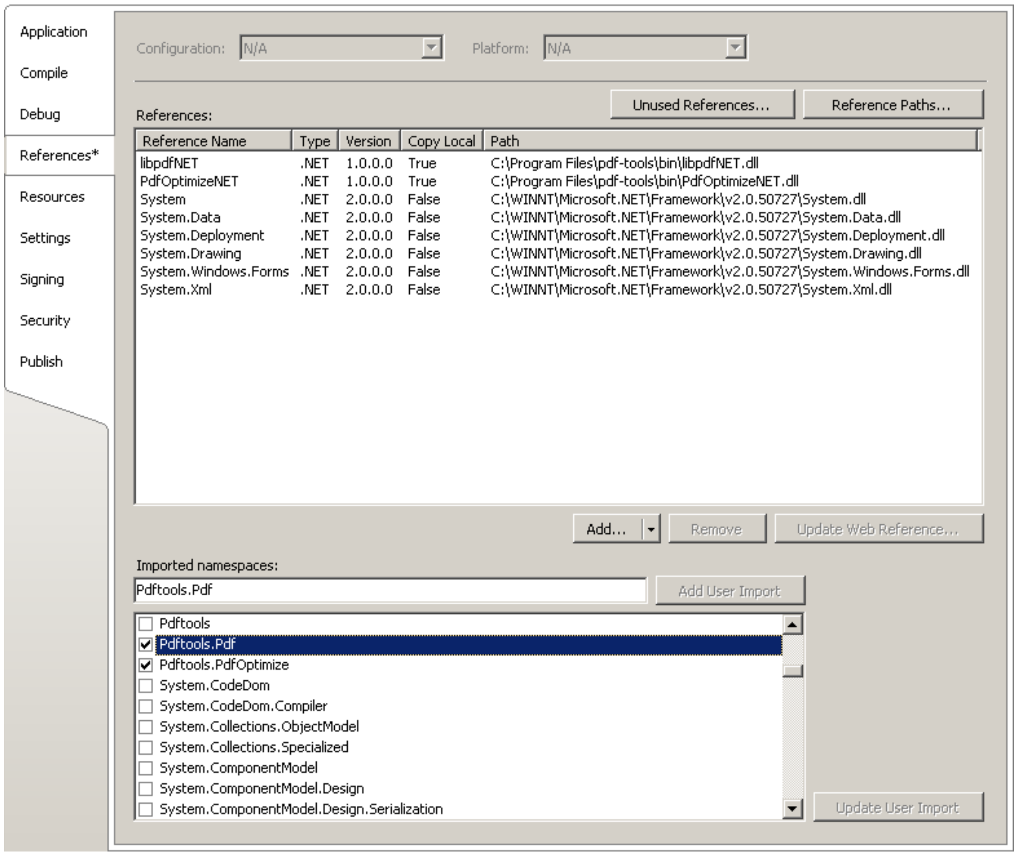
Steps 3 and 4 are shown separately for C# and Visual Basic.

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 two namespaces *Pdftools.Pdf* and *Pdftools.PdfSecure*.

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



Configuration: N/A Platform: N/A

References:

Reference Name	Type	Version	Copy Local	Path
libpdfNET	.NET	1.0.0.0	True	C:\Program Files\pdf-tools\bin\libpdfNET.dll
PdfOptimizeNET	.NET	1.0.0.0	True	C:\Program Files\pdf-tools\bin\PdfOptimizeNET.dll
System	.NET	2.0.0.0	False	C:\WINNT\Microsoft.NET\Framework\v2.0.50727\System.dll
System.Data	.NET	2.0.0.0	False	C:\WINNT\Microsoft.NET\Framework\v2.0.50727\System.Data.dll
System.Deployment	.NET	2.0.0.0	False	C:\WINNT\Microsoft.NET\Framework\v2.0.50727\System.Deployment.dll
System.Drawing	.NET	2.0.0.0	False	C:\WINNT\Microsoft.NET\Framework\v2.0.50727\System.Drawing.dll
System.Windows.Forms	.NET	2.0.0.0	False	C:\WINNT\Microsoft.NET\Framework\v2.0.50727\System.Windows.Forms.dll
System.Xml	.NET	2.0.0.0	False	C:\WINNT\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll

Imported namespaces:

Pdftools.Pdf

- ☐ Pdftools
- ☑ Pdftools.Pdf
- ☑ Pdftools.PdfOptimize
- ☐ System.CodeDom
- ☐ System.CodeDom.Compiler
- ☐ System.Collections.ObjectModel
- ☐ System.Collections.Specialized
- ☐ System.ComponentModel
- ☐ System.ComponentModel.Design
- ☐ System.ComponentModel.Design.Serialization

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

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

C#

3. Add the following namespaces:

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

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

```
Optimizer opt = new Optimizer ();
opt.Open(...);
...
opt.Close();
```

Troubleshooting

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

To resolve that ensure the native DLL is found at execution time. For this, see sub-chapter ".NET Interface" in the chapter "Installation".

7 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.

In this documentation it is distinguished between different types of images.

Bi-tonal: an image that consists of only black and white pixels

Monochrome: an image that has 1 channel (8 bit grayscale)

Color: a continuous-tone image that has 3 or 4 channels (24 bit RGB or 32 bit CMYK)

7.1 Methods

Close

```
Method Boolean Close()
```

Close an opened input file.

- Return value:

  True: The file was closed successfully.

  False: Otherwise

GetPDF

```
Method Variant GetPDF()
```

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

- Return value: A byte array containing the optimized PDF. In certain programming languages, such as Visual Basic 6, the type of the byte array must explicitly be Variant.

LinarizeFile

```
Method Boolean LinarizeFile (String FileName, String Password, String String
OutFileName, String UserPw, String OwnerPw, Long PermissionFlags)
```

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

- Parameters

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

  Password (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 an be passed as argument.

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

  UserPw (optional): The user password of the output PDF file.

  OwnerPw (optional): The owner password of the output PDF file.

  PermissionFlags (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*.

- Return value:

  True: A linearized PDF file was successfully created.

  False: Otherwise

ListFonts

Method Boolean ListFonts(String FileName)

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

- Parameters:

  FileName: 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	This file name if 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*

- Return values:

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

  False: Otherwise

ListImages

Method Boolean ListImages(String FileName)

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

- Parameters:

  FileName: 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 inches, 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	This file name if 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.*

- Return values:

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

  False: Otherwise

Open

Method Boolean Open(String FileName, String Password)

Open a PDF random access disk file, i.e. make the objects contained in the PDF document accessible. If the document is already open it is closed first.

- Parameters:

  FileName: The file name and optionally the file-path, drive or server string according to the operating systems file name specification rules.

  Password (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.

- Return value:

  True: The file could successfully be opened.

  False: The file does not exist, it is corrupt, or the password is invalid.

OpenMem

```
Method Boolean OpenMem(Variant varMem, String Password)
```

Open a PDF memory-block, i.e. make the objects contained in the PDF document accessible. If the document is already open it is closed first.

- Parameters:

  varMem: The memory block containing the PDF file given as a one dimensional byte array.

  Password (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.

- Return value:

  True: The file could successfully be opened.

  False: The file does not exist, it is corrupt, or the password is invalid.

RenameFont

Deprecated

SaveAs

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

Create an output PDF document, optimizes the input file and saves it to the new file.

- Parameters:

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

  UserPwd (optional): Set the user password of the PDF document. If this parameter is omitted, the default password is used. Use 0 to set no password.

  OwnerPwd (optional): Set the owner password of the PDF document. If this parameter is omitted, the default password is used. Use 0 to set no password.

  PermissionFlags (optional): The permission flags. By default no permissions are granted. The permissions that can be granted are listed at the enumeration *TPDFPermissionFlags*.

  To not encrypt the output document, set PermissionFlags to –1, user and owner password to 0.

  In order to allow high quality printing, flags *ePermPrint* and *ePermDigitalPrint* need to be set.

- Return value:

  True: The optimized document could successfully be saved to file.

  False: Otherwise

SaveInMemory

```
Method Boolean SaveInMemory()
```

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

- Return value:

  True: The optimized document could successfully be saved in memory.

  False: Otherwise

SetInfoEntry

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

Set the value of an info entry key. Examples for keys are “Author”, “Subject”, “Title” or custom attributes.

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”[2]. There are three parameters that influence the version of the PDF output file:

- The value set in this property
- The PDF version of the input file
- Other settings in the optimization (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.

Examples

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

   ```
   SetVersion(“1.4”)
   ```

   The output file is PDF version 1.5.

2. 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.

3. 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.

7.2 Properties

BitonalCompression

```
Property TPDFCompression BitonalCompression
Accessors: Get, Set
Default: eComprGroup4
```

Deprecated, use *BitonalCompressions*.

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

BitonalCompressions

```
Property TPDFComprAttempt BitonalCompressions
Accessors: Get, Set
Default: eComprGroup4
```

Get or set the compression types for bi-tonal images. Typically, CCITT G4 or JBIG2 is used for bi-tonal compression. Due to the simpler algorithm CCITT G4 has the advantage of being faster. JBIG2 can achieve compression ratios that are up to twice as high as CCITT G4 at the cost of longer computation time. See also enumeration *TPDFComprAttempt*.

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

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

Others values are ignored.

BitonalResolutionDPI

```
Property Float BitonalResolutionDPI
Accessors: Get, Set
Default: 150
```

Get or set the target resolution in dots per inch (DPI) for bi-tonal images.

BitonalThresholdDPI

```
Property Float BitonalThresholdDPI
Accessors: Get, Set
Default: 225
```

Get or set the threshold resolution in DPI for bi-tonal images. See also *ThresholdDPI*.

ClipImages

```
Property Boolean ClipImages
Accessors: Get, Set
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.

ColorCompression

```
Property TPDFCompression ColorCompression
Accessors: Get, Set
Default: eComprJPEG
```

Deprecated, see *ContinuousCompressions*.

ColorConversion

```
Property TPDFColorConversion ColorConversion
Accessors: Get, Set
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.

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*.

ColorResolutionDPI

```
Property Float ColorResolutionDPI
Accessors: Get, Set
Default: 150
```

Get or set the target resolution in dots per inch (DPI) for color images.

ColorThresholdDPI

```
Property Float ColorThresholdDPI
Accessors: Get, Set
Default: 225
```

Get or set the threshold resolution in DPI for color images. See also *ThresholdDPI*.

ContinuousCompressions

```
Property TPDFComprAttempt ContinuousCompressions
Accessors: Get, Set
Default: eComprAttemptJPEG
```

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:

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

Others values are ignored.

CompressionQuality

Deprecated, use *ImageQuality* instead.

ConvertToCFF

```
Property Boolean ConvertToCFF
Accessors: Get, Set
Default: False
```

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

ErrorCode

```
Property TPDFErrorCode ErrorCode
Accessors: Get
```

Get the error code of the last operation. See enumeration *TPDFErrorCode*.

ExtractFonts

```
Property Boolean ExtractFonts
Accessors: Get, Set
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.

ExtractImages

```
Property Boolean ExtractImages
Accessors: Get, Set
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.

ForceRecompression

```
Property Boolean ForceRecompression
Accessors: Get, Set
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.

ImageStratConserv

```
Property Boolean ImageStratConserv
Accessors: Get, Set
Default: false
```

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 (*ThresholdDPI*, *ThresholdDPI*), or if it has undergone a color conversion (*ColorConversion*), otherwise the original input image is taken as is. See also *Optimizing Raster Images*.

ImageQuality

```
Property Single ImageQuality
Accessors: Get, Set
Default: 75
```

Get or set the quality index of the lossy compression. This is a value from 1 to 100. This can be applied for JPEG, JPEG2000 and JBIG2 compression. For JBIG2 only the values from 10 to 100 that are multiples of 10 are supported. For both JPEG2000 and JBIG2, a quality index of 100 means lossless compression. JPEG compression is always lossy.

IndexedCompressions

```
Property TPDFComprAttempt IndexedCompressions
Accessors: Get, Set
Default: eComprAttemptFlate
```

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

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

- `eComprAttemptRaw`
- `eComprAttemptFlate`
- `eComprAttemptLZW`
- `eComprAttemptSource`

Others values are ignored.

Linearize

```
Property Boolean Linearize
Accessors: Get, Set
Default: False
```

Get or set whether to linearize the PDF output file for fast web access.

Linearization is the process of preparing a PDF file in a way that permits random page access by a web browser. While the whole non-linearized PDF file must be downloaded before the first page can be displayed, this is not the case for a linearized file.

MrcLayerCompression

```
Property TPDFCompression MrcLayerCompression
Accessors: Get, Set
Default: eComprJPEG
```

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*.

MrcLayerQuality

```
Property Short MrcLayerQuality
Accessors: Get, Set
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*.

MrcLayerResolutionDPI

```
Property Float MrcLayerResolutionDPI
Accessors: Get, Set
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 *Optimizing Raster Images* and *Mixed Raster Content (MRC) Optimization for Images*.

MrcMaskCompression

```
Property TPDFCompression MrcMaskCompression
Accessors: Get, Set
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*.

MrcPictCompression

```
Property TPDFCompression MrcPictCompression
Accessors: Get, Set
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*.

MergeEmbeddedFonts

```
Property Boolean MergeEmbeddedFonts
Accessors: Get, Set
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.

MonochromeCompression

```
Property TPDFCompression MonochromeCompression
Accessors: Get, Set
Default: eComprJPEG
```

Deprecated, see *ContinuousCompressions*.

MonochromeResolutionDPI

```
Property Float MonochromeResolutionDPI
Accessors: Get, Set
Default: 150
```

Get or set the resolution in DPI for monochrome images.

MonochromeThresholdDPI

```
Property Float MonochromeThresholdDPI
Accessors: Get, Set
Default: 150
```

Set or get the threshold resolution dpi for monochrome images. See also *ThresholdDPI*.

OptimizeResources

```
Property Boolean OptimizeResources
Accessors: Get, Set
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.

PageCount

```
Property Long PageCount
Accessors: Get
```

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

RemoveNonSymbolicFonts

```
Property Boolean RemoveNonSymbolicFonts
Accessors: Get, Set
Default: False
```

Get or set whether non-symbolic fonts should be removed. If a font has no Unicode information, the font is not removed, and remains embedded instead. See the property RemoveStandardFonts for more information on the importance of Unicode information when un-embedding fonts.

What is a symbolic font?

A symbolic font contains non standard character sets. Font programs of symbolic fonts have encodings that are usually built-in and unique to each font. Two of the 14 PDF Standard Fonts are symbolic: Symbol and ZapfDingbats.

RemoveRedundantObjects

```
Property Boolean RemoveRedundantObjects
Accessors: Get, Set
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.

RemoveStandardFonts

```
Property Boolean RemoveStandardFonts
Accessors: Get, Set
Default: False
```

Get or set whether to remove all embedded standard fonts and replace them with one of the 14 PDF Standard Fonts. The following font families are considered standard fonts:

Arial	*CourierNewPS*	*Times*	*ZapfDingbats*
Courier	*Helvetica*	*TimesNewRoman*	
CourierNew	*Symbol*	*TimesNewRomanPS*	

And their derivatives (they are different for different font families) such as:

Arial,Bold	*Arial-Bold*	*Arial-Italic*	*ArialMT*
Arial,BoldItalic	*Arial-BoldItalic*	*Arial-BoldMT*	*Courier-Bold*
Arial,Italic	*Arial-BoldItalicMT*	*Arial-ItalicMT*	*Courier-Oblique*

A PDF Viewer must be able to display standard fonts correctly, even if they are not embedded. Therefore using this option should not visually alter the PDF when it is displayed. Un-embedding a font decreases the file size.

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. 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 un-embedding of the font will not lead to visual differences.

ResolutionDPI

```
Property Single ResolutionDPI
Accessors: Get, Set
Default: 150
```

Get or set the resolution in dpi after re-sampling images. This property affects all three image compression types (bi-tonal, monochrome, color). The 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.

Strip

```
Property TPDFStripType Strip
Accessors: Get, Set
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. Several values of *TPDFStripType* can be combined with the bitwise or operator.

SubsetFonts

```
Property Boolean SubsetFont
Accessors: Get, Set
Default: false
```

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.

ThresholdDPI

```
Property Single ThresholdDPI
Accessors: Get, Set
Default: 225
```

Get the threshold in dpi to selectively activate re-sampling. Only images with a resolution above the threshold dpi will be re-sampled. The typical threshold value when optimizing for the web is 225 dpi (default). This property affects all three image compression types (bi-tonal, monochrome, color). Set to -1 to deactivate re-sampling.

7.3 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).

TPDFColorConversion

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

TPDFCompression

Compression types as occurring in PDF. Note that not all image formats/color depths support all compression types.
See also chapter *Supported Image Compression Types*.

ECompRaw	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

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.

EComprAttemptRaw	No compression
eComprAttemptJPEG	Joint Photographic Expert Group
eComprAttemptFlate	Flate compression
eComprAttemptLZW	Lempel-Ziv-Welch
eComprAttemptGroup3	CCITT Fax Group 3
eComprAttemptGroup3_2D	CCITT Fax Group 3 2D
eComprAttemptGroup4	CCITT Fax Group 4
eComprAttemptJBIG2	Joint Bi-level Image Experts Group
eComprAttemptJPEG2000	JPEG2000
eComprAttemptMRC	Perform mixed raster content analysis
eComprAttemptSource	Same compression type as the original image

TPDFErrorCode

All TPDFErrorCode enumerations start with “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 in an indication of the type of error. These are: **S**uccess, **E**rror, **W**arning and **I**nformation. With respect to corrupt PDF files: An error indicates a corruption in the PDF, the file may or may not be readable. A warning indicates the file is readable but not valid.

A full list of all PDF Tools error codes is available in the header file *pdferror.h*. The error codes that are listed to file access are listed here.

PDF_S_SUCCESS	The operation was completed successfully.
PDF_E_EVAL	This software is an evaluation version. Please contact www.pdf-tools.com.
PDF_E_FILEOPEN	File open failed.
PDF_E_FILECREATE	Create file failed.
PDF_E_PASSWORD	The authentication failed due to a wrong password.

TPDFFontType

eFontType1	Type 1 Font
eFontTrueType	True Type Font

eFontCFF	Compact Font Format
eFontType3	Type 3 Font

TPDFPermission

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

ePermPrint	Low resolution printing
ePermModify	Changing the document
ePermCopy	Content copying or extraction
ePermAnnotate	Annotations
ePermFillForms	Filling of form fields
ePermSupportDisabilities	Support for disabilities
ePermAssemble	Document Assembly
ePermDigitalPrint	High resolution printing
ePermAll	Equivalent to setting all the above flags. The output file is, however, still encrypted.
ePermNoEncryption	If this is the only flag set then the output file will not be encrypted.

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`**

TPDFStripType

eStripThreads	Strip thumbnails
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
eStripFormsAnnots	Strip and flatten form fields and annotations (This removes interactive features of the PDF.)
eStripAll	Strip everything (all of the above)

7.4 Supported Image Compression Types

For additional information about compressions in PDF, see also ISO 32000, chapter 7.4.

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	Group 3: 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 if the image quality index is set to 100 Lossy otherwise
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 doesn't 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 G3 or G4 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	JPEG 2000 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)

8 Samples

8.1 Suggested Values to Optimize for the Web

```
ColorConversion = eConvRGB
BitonalCompressions = eComprAttemptGroup4 Or eComprAttemptJBIG2 Or
eComprAttemptSource
ContinuousCompressions = eComprAttemptJPEG Or eComprAttemptJPEG2000 Or
eComprAttemptSource
IndexedCompressions = eComprAttemptFlate Or eComprAttemptSource
ClipImages = True
OptimizeResources = True
RemoveRedundantObjects = True
SubsetFonts = True
MergeEmbeddedFonts = True
RemoveStandardFonts = True
Linearize = True
```

Additionally, optional information can be stripped to further minimize the file size:

```
Strip = eStripThreads Or eStripMetadata Or eStripPieceInfo Or eStripStructTree
Or eStripThumb Or eStripSpider Or eStripAlternates
```

If encrypting:

```
SaveAs(<filename>, "", <ownerpassword>, ePermPrint Or ePermFillForms)
```

8.2 Suggested Values for Printing

```
ColorConversion = eConvCMYK
ThresholdDPI = -1
RemoveRedundantObjects = True
OptimizeResources = True
MergeEmbeddedFonts = True
SubsetFonts = True
```

Additionally, optional information can be stripped to further minimize the file size:

```
Strip = eStripThreads Or eStripMetadata Or eStripPieceInfo Or eStripStructTree
Or eStripThumb Or eStripSpider Or eStripFormsAnnots
```

If encrypting:

```
SaveAs(<filename>, "", <ownerpassword>, ePermPrint Or ePermDigitalPrint)
```