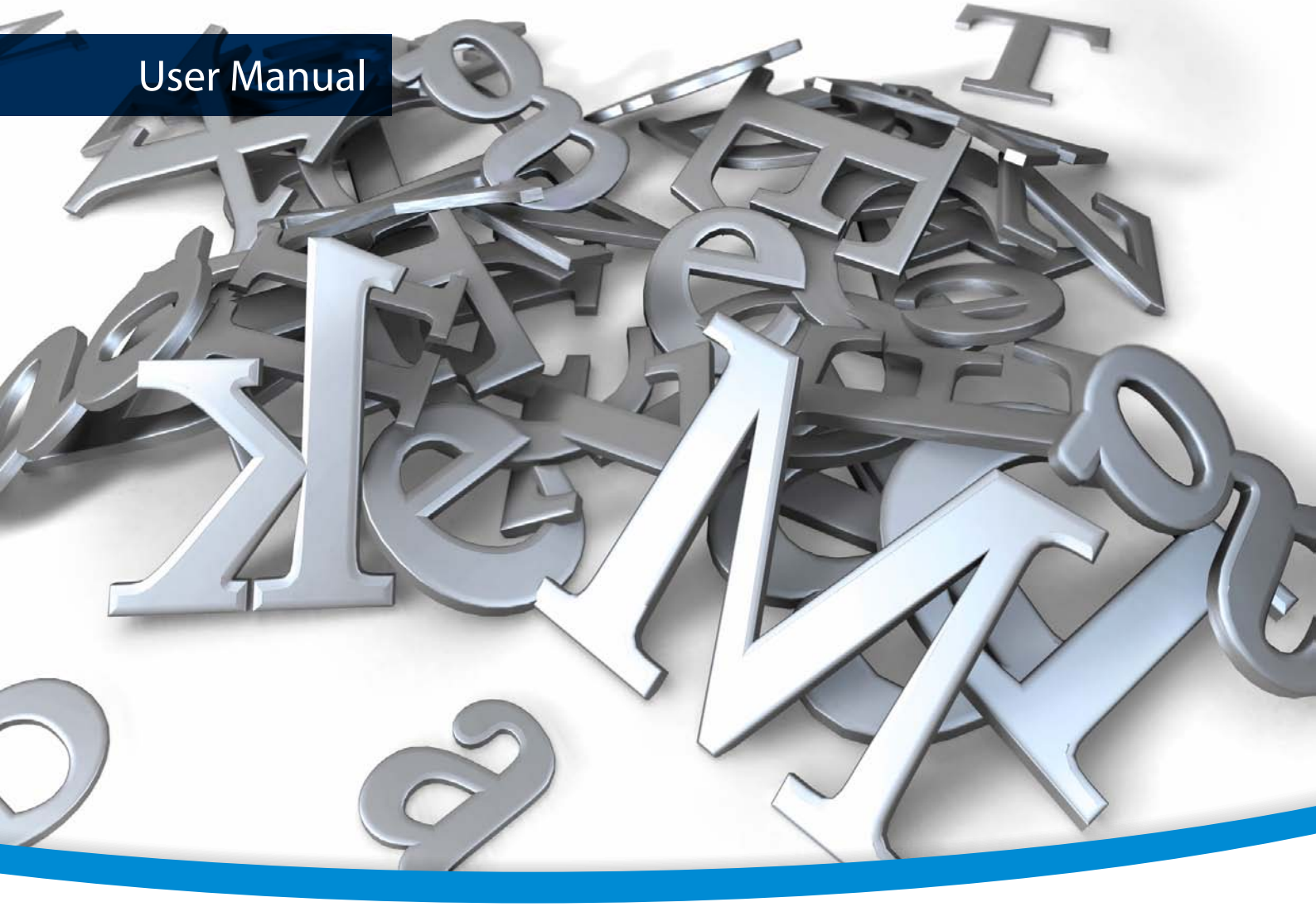
User Manual



3-Heights®
PDF OCR API

PDF/A conform

Version 6.27.8

PDF-TOOLS.COM
Premium PDF Technology

Contents

1 Introduction

1.1 Description

The 3-Heights® PDF OCR API enhances PDF documents using information detected by an OCR engine.

All text in PDF documents can be made extractable, regardless of how text is included in the document. Specifically, text in images, text written with vector graphics or other graphical effects (e.g. transparency effects), or text using a font that does not provide extractable text (i.e. Unicode information) can be detected.

Tagging of OCR text for accessibility is supported. This is useful as preparation for PDF/A level A conversion or to process tagged documents.

Detected barcodes or QR codes can be extracted or embedded into the document's metadata.

The product is optimized for performance, which guarantees low latency and high document throughput. This is achieving by minimizing the number of required OCR operations. Furthermore, OCR operations are executed asynchronously, if supported by the OCR engine.

1.2 Functions

1.2.1 Features

- Make text extractable
  - Text contained in images
  - Text with fonts that have no Unicode information
  - Text written using vector graphics (e.g. in CAD drawings)
  - Any visible text, regardless of the type of graphics objects used
- Scan improvements
  - Deskew scanned images
  - Rotate pages according to the recognized rotation of scan
- Detect barcodes and QR codes
- Process embedded files
- Tagging of OCR text for accessibility
- High performance
  - Asynchronous processing
  - Page analysis and result caching to minimize OCR operations
- High quality
  - Conform to PDF/A
  - High-fidelity conversion of existing page content
  - 3-Heights® PDF Rendering Engine 2.0.
  - Automatic detection of optimal OCR resolution

1.2.2 Formats

- PDF 1.x (PDF 1.0, …, PDF 1.7)
- PDF 2.0
- PDF/A-1, PDF/A-2, PDF/A-3

1.2.3 Conformance

Standards:

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

1.3 Interfaces

The following interfaces are available:

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

1.4 Operating systems

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

- Windows Client 7+ | x86 and x64
- Windows Server 2008, 2008 R2, 2012, 2012 R2, 2016, 2019, 2022 | x86 and x64
- Linux:
  - Red Hat, CentOS, Oracle Linux 7+ | x64
  - Fedora 29+ | x64
  - Debian 8+ | x64
  - Other: Linux kernel 2.6+, GCC toolset 4.8+ | x64

'+' indicates the minimum supported version.

1.5 How to best read this manual

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

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

[1] Limited supported OS versions. Operating systems

2 Installation and deployment

2.1 Windows

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

To install the software, proceed as follows:

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

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

4. The usage of the NuGet package is described in section NuGet package.
5. (Optional) Register your license key using the License management.
6. Identify the interface you are using. Perform the specific installation steps for that interface described in Interface-specific installation steps.
7. Make sure your platform meets the requirements regarding fonts described in Fonts.
8. Download and install the 3-Heights® OCR Service, the OCR Service client plugin,, and the OCR Engine as described in the respective manuals:
   - 3-Heights® OCR Add-on for ABBYY FineReader Engine v10: `OcrAbbyy10.pdf`
   - 3-Heights® OCR Add-on for ABBYY FineReader Engine v11: `OcrAbbyy11.pdf`
   - 3-Heights® OCR Add-on for ABBYY FineReader Engine v12: `OcrAbbyy12.pdf`
   - 3-Heights® OCR Add-on for Barcode and QR Code Recognition: `OcrBarcodes.pdf`
   - 3-Heights® OCR Service: `OcrService.pdf` from the separate product kit.

2.2 Linux and macOS

This section describes installation steps required on Linux or macOS.

The Linux and macOS version of the 3-Heights® PDF OCR API provides two interfaces:

- Java interface
- Native C interface

Here is an overview of the files that come with the 3-Heights® PDF OCR API:

File description

Name	Description
`bin/x64/libPdfOcrAPI.so`	Shared library that contains the main functionality. The file's extension differs on macOS, (`.dylib` instead of `.so`).
`bin/x64/*.ocr`	OCR plugin modules
`doc/*.*`	Documentation
`include/*.h`	Header files to include in your C/C++ project
`jar/PdfOcrAPI.jar`	Java API archive
`samples`	Example code

2.2.1 Linux

1. Unpack the archive in an installation directory, e.g. `/opt/pdf-tools.com/`
2. Verify that the GNU shared libraries required by the product are available on your system:

   ```
   ldd libPdfOcrAPI.so
   ```

   If the previous step reports any missing libraries, you have two options:
   a. Download an archive that is linked to a different version of the GNU shared libraries and verify whether they are available on your system. Use any version whose requirements are met. Note that this option is not available for all platforms.
   b. Use your system's package manager to install the missing libraries. It usually suffices to install the package `libstdc++6`.
3. Create a link to the shared library from one of the standard library directories, e.g.

   ```
   ln -s /opt/pdf-tools.com/bin/x64/libPdfOcrAPI.so /usr/lib
   ```

4. Optionally, register your license key using the license manager.
5. Identify the interface you are using. Perform the specific installation steps for that interface described in Interface-specific installation steps.
6. Make sure your platform meets the requirements regarding fonts described in Fonts.
7. Download and install the 3-Heights® OCR Service, the OCR Service client plugin,, and the OCR Engine as described in the respective manuals:
   - 3-Heights® OCR Add-on for ABBYY FineReader Engine v10: `OcrAbbyy10.pdf`
   - 3-Heights® OCR Add-on for ABBYY FineReader Engine v11: `OcrAbbyy11.pdf`
   - 3-Heights® OCR Add-on for ABBYY FineReader Engine v12: `OcrAbbyy12.pdf`

- 3-Heights® OCR Add-on for Barcode and QR Code Recognition: OcrBarcodes.pdf
- 3-Heights® OCR Service: OcrService.pdf from the separate product kit.

2.3 ZIP archive

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

Interface	Programming languages
.NET	The MS software platform .NET can be used with any .NET capable programming language such as: ■ C# ■ VB .NET ■ J# ■ others For a convenient way to use this interface, see NuGet package.
Java	The Java interface is available on all platforms.
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 development kit (SDK) contains all files that are used for developing the software. The role of each file in each of the four different interfaces is shown in table Files for development. The files are split in four categories:

Req. The file is required for this interface.

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

Doc. The file is for documentation only.

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

Files for development

Name	.NET	Java	C
bin\‹platform›\PdfOcrAPI.dll	Req.	Req.	Req.
bin*NET.dll	Req.		
bin*NET.xml	Doc.		
bin\‹platform›*.ocr	Req.	Req.	Req.
doc*.pdf	Doc.	Doc.	Doc.
doc\javadoc*.*		Doc.	

Files for development

Name	.NET	Java	C
include\pdfocrapi_c.h			Req.
include*.*			Opt.
jar\PdfOcrAPI.jar		Req.	
lib\‹platform›\PdfOcrAPI.lib			Req.[2]
samples*.*	Doc.	Doc.	Doc.

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

File description

Name	Description
bin\‹platform›\PdfOcrAPI.dll	DLL that contains the main functionality (required), where ‹platform› is either Win32 or x64 for the 23-bit or the 64-bit library, respectively.
bin*NET.dll	.NET assemblies are required when using the .NET interface. The files bin*NET.xml contain the corresponding XML documentation for MS Visual Studio.
bin\‹platform›*.ocr	OCR plugin DLLs[3]
doc*.*	Documentation
include*.*	Files to include in your C / C++ project
lib\‹platform›\PdfOcrAPI.lib	On Windows operating systems, the object file library needs to be linked to the C/C++ project.
jar\PdfOcrAPI.jar	Java API archive
samples*.*	Sample programs in different programming languages

2.3.2 Deployment

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

[2] Not required for Linux or macOS.
[3] These files must reside in the same directory as PdfOcrAPI.dll.

Files for deployment

Name	.NET	Java	C
bin\‹platform›\PdfOcrAPI.dll	Req.	Req.	Req.
bin*NET.dll	Req.		
bin\‹platform›*.ocr	Req.	Req.	Req.
jar\PdfOcrAPI.jar		Req.	

The deployment of an application works as described below:

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

2.4 NuGet package

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

Installation

The package `PdfTools.PdfOcr 6.27.8` is available on nuget.org. Right-click on your .NET project in Visual Studio and select "Manage NuGet Packages...". Finally, select the package source "nuget.org" and navigate to the package `PdfTools.PdfOcr 6.27.8`.

Development

The package `PdfTools.PdfOcr 6.27.8` contains .NET libraries with versions .NET Standard 1.1, .NET Standard 2.0, and .NET Framework 2.0, and native libraries for Windows, and Linux.

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

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

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

2.5 Interface-specific installation steps

2.5.1 Java interface

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

For compilation and execution

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

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

For execution

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

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

On Linux or macOS, the path separator usually is a colon and hence the above changes to something like:

```
... -classpath ".:/path/to/PdfOcrAPI.jar" ...
```

2.5.2 .NET interface

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

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

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

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

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

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

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

Troubleshooting: TypeInitializationException

The most common issue when using the .NET interface is that the correct native DLL `PdfOcrAPI.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.

[4] On Windows defined by the environment variable PATH, and on Linux defined by `LD_LIBRARY_PATH`.
[5] If the wrong data model is used, there is an error message similar to this: "`Can't load IA 32-bit .dll on a AMD 64-bit platform`"

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

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

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

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

Troubleshooting: DllNotFoundException

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

Resolve this by performing one of these actions:

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

Troubleshooting: BadImageFormatException

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

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

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

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

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

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

2.5.3 C interface

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

2.6 Uninstall, Install a new version

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

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

2.7 Fonts

Fonts are required, if OCR is preformed and OCR text is added to a PDF document. Therefore, it is crucial, that the fonts available in the Font directories contain all characters required for the OCR text. For example, when recognizing Japanese OCR text, it is recommended to add the fonts "MS Mincho" or "MS Gothic" to the Font directories.

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

On Linux and macOS, it is recommended to install the Liberation fonts, Google Noto CJK fonts, and the OpenSymbol font. On Debian based systems, the packates are called `fonts-liberation2`, `fonts-noto-cjk`, and `fonts-opensymbol`.

2.7.1 Font cache

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

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

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

2.7.2 Microsoft core fonts on Linux or macOS

Many PDF documents use Microsoft core fonts like Arial, Times New Roman, and other fonts commonly used on Windows. Therefore, it is recommended to install these fonts to your default font directories. Many Linux distributions offer an installable package for these "Microsoft TrueType core fonts". For instance, on Debian based systems, the package is called `ttf-mscorefonts-installer`.

Alternatively, you can download the fonts from here:

`https://corefonts.sourceforge.net/`

Microsoft has an FAQ on the subject, that covers licensing related questions as well:

`https://docs.microsoft.com/en-us/typography/fonts/font-faq`

2.8 Note about the evaluation license

With the evaluation license, the 3-Heights® PDF OCR API automatically adds a watermark to the output files.

2.9 Special directories

2.9.1 Directory for temporary files

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

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

Windows

1. The path specified by the `%TMP%` environment variable
2. The path specified by the `%TEMP%` environment variable
3. The path specified by the `%USERPROFILE%` environment variable
4. The Windows directory

Linux and macOS

1. The path specified by the `$PDFTMPDIR` environment variable
2. The path specified by the `$TMP` environment variable
3. The `/tmp` directory

2.9.2 Cache directory

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

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

Windows

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

Linux and macOS

- If the user has a home directory:
  `~/.pdf-tools/Caches`
- If the user has no home directory:
  `<TempDirectory>/pdf-tools/Caches`

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

2.9.3 Font directories

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

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

Windows

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

Linux

1. `/usr/share/fonts`
2. `/usr/local/share/fonts`
3. `~/.fonts`
4. `$PDFFONTDIR` or `/usr/lib/X11/fonts/Type1`

3 License management

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

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

4 User guide

4.1 Overview of the API

4.1.1 Process description

The following is a simplified process description of the 3-Heights® PDF OCR API:

1. **Open document**: The input document is opened.
2. **Process document**: The document is processed and the result written to the output.
    1. **Process pages**
        1. Analyze page: The page's content is analyzed in order to determine, whether processing by the OCR engine is required or not (see chapter Page analysis below).
        2. OCR page (optional)
            1. Determine optimal OCR resolution
            2. Render page: The page is converted to an image using the 3-Heights® PDF Rendering Engine 2.0.
            3. Send image to OCR engine.
            4. Process OCR results (see chapter OCR result processing below).
        3. Copy page: The page is copied to the output. If available, information from the OCR engine is added.
    2. **Process embedded files**: Embedded files can be processed recursively or copied as-is.

Page analysis

The page's content is analyzed. The modes of ocr parameters specified for images (`ImageOcrParams`), text (`TextOcrParams`), and pages (`PageOcrParams`) are evaluated. The page is processed by the OCR engine if required by any of the modes.

OCR result processing

Each OCR result object is processed as follows:

1. If it is a barcode, it is processed according to the `BarcodeMode`.
2. If its location is on an image on the page, it is processed according to the `ImageOcrMode`.
3. If it corresponds to text on the page, it is processed according to the `TextOcrMode`.
4. Otherwise it is added as OCR text to the page, if the `PageOcrMode` is not `None`.

4.1.2 Example

Example: C# example that shows how to add OCR text to images.

```
void OcrProcessImages(Stream input, Stream output)
{
    // Open input document from stream
    using (var document = Document.Open(input, null))
    {
        // Create OCR engine
        using (var engine = Engine.Create(engineName))
        {
            // Configure OCR engine
```

```
            engine.SetParameters(engineParams);
            engine.SetLanguages(engineLanguages);

            // Set process parameters
            var ocr = new OcrParams();
            ocr.Engine = engine;

            var imageOcr = new ImageOcrParams();
            imageOcr.Mode = ImageOcrMode.UpdateText;

            // Process document and write result to output stream
            document.Process(output, null, ocr, imageOcr,
                             null, null, null);
        }
    }
}
```

4.2 Use cases

This chapter describes some common use cases.

4.2.1 How to make text extractable

This example shows how text in a PDF document can be made extractable. This is suitable both for born-digital and scanned documents. Fully functional C# and Java samples are contained in the product kit or can be downloaded from the product website.

> **Note:** For tagged input documents, the configuration described in How to tag scans for accessibility (PDF/A level A) should be used.

OCR engine configuration

Abbyy FineReader 11 or 12

The following profile configuration `abbyy_text.ini` is optimized to extract as much text as possible:

```
[PagePreprocessingParams]
CorrectOrientation=TRUE
[PageAnalysisParams]
DetectVerticalEuropeanText=TRUE
[ObjectsExtractionParams]
DetectTextOnPictures = TRUE
```

Use the profile using the OCR engine running on the OCR Service:

```
using (var engine = Engine.Create("service"))
{
    engine.SetParameters(@"Profile=C:\path\to\abbyy_text.ini");
    engine.SetLanguages(languages);
    ...
```

```
}
```

Note: It is important that `C:\path\to\abbyy_text.ini` is the path to the configuration file on the OCR Service and the OCR Service process has read permissions.

Processing configuration

1. **Detect text contained in images**: For documents that contain images, processing of images can be activated by setting an image OCR mode (see `ImageOcrMode`):

```
var imageOcr = new ImageOcrParams();
imageOcr.Mode = ImageOcrMode.UpdateText;
```

2. **Make text extractable**: For documents that contain non-extractable text, processing of text can be activated by setting a text OCR mode (see `TextOcrMode`):

```
var textOcr = new TextOcrParams();
textOcr.Mode = TextOcrMode.Update;
```

3. **Make other visible text extractable**: For documents that contain other forms of visible text, pages can be OCR processed by setting a page OCR mode (see `PageOcrMode`):

```
var pageOcr = new PageOcrParams();
pageOcr.Mode = PageOcrMode.IfNoText;
```

Process document

```
var ocr = new OcrParams();
ocr.Engine = engine;

var warnings = document.Process(output, null, ocr, imageOcr,
                                textOcr, pageOcr, null);
```

4.2.2 How to tag scans for accessibility (PDF/A level A)

"Tagging" adds structural information to a PDF. This information can be used e.g. to read the document to the visually impaired.

The 3-Heights® PDF OCR API supports tagging scans, e.g. such that they can be converted to PDF/A level A.

Tagging of scans that contain no figures or pictures, will conform to the PDF/UA specification (ISO 14289-1). For figures and pictures an alternative representation or replacement text must be included. Because this information is not provided by the OCR engine, tagging of such files cannot conform to PDF/UA.

Prerequisites

1. The OCR engine must provide structural information for OCR results. We recommend Abbyy FineReader 11 or newer.
2. If the 3-Heights® OCR Service is used, it must be newer than 4.11.21.0.

OCR engine configuration

Abbyy FineReader 11 or 12

The following profile configuration `abbyy_tagging.ini` is suitable:

```
[PagePreprocessingParams]
CorrectOrientation=TRUE
```

This is essentially the predefined profile "DocumentConversion_Accuracy", but can also handle rotated pages.

Use the profile using the OCR engine running on the OCR Service:

```
using (var engine = Engine.Create("service"))
{
    engine.SetParameters(@"Profile=C:\path\to\abbyy_tagging.ini");
    engine.SetLanguages(languages);
    ...
}
```

> **Note:** It is important that `C:\path\to\abbyy_tagging.ini` is the path to the configuration file on the OCR Service and the OCR Service process has read permissions.

Processing configuration

1. Activate processing of images by setting an image OCR mode:

```
var imageOcr = new ImageOcrParams();
imageOcr.Mode = ImageOcrMode.UpdateText;
```

2. (Optional) Enable scan enhancements:

```
imageOcr.RotateScan = true;
imageOcr.DeskewScan = true;
```

3. Activate creation of tagging information by setting the tagging mode:

```
var pageOcr = new PageOcrParams();
pageOcr.Tagging = TaggingMode.Update;
```

Process document

```
var ocr = new OcrParams();
ocr.Engine = engine;

var warnings = document.Process(output, null, ocr, imageOcr,
                                null, pageOcr, null);
```

Error handling

Tagging errors are classified as warnings by the 3-Heights® PDF OCR API. Because tagging is crucial for this process, tagging warnings returned by `Process` must be checked and treated as errors.

```
foreach (var warning in warnings)
{
    if (warning.Code == WarningCode.Tagging)
        throw new Exception(warning.Message);
}
```

See How to handle conversion warnings for more information.

4.2.3 How to detect barcodes

This example shows how to detect and extract barcodes and QR codes. Fully functional C# and Java samples of this use case are contained in the product kit or can be downloaded from the product website.

OCR engine configuration

There are two OCR engines available that support barcode recognition.

Barcodes OCR engine

The OCR engine "`barcodes`" is a specialized plugin for barcode and QR code recognition.

No engine parameters are required. However, in order to speed up the recognition process, it can be limited to a specific set of code types:

```
using (var engine = Engine.Create("barcodes"))
{
    engine.SetParameters(@"BarcodeTypes=QRCode");
    ...
}
```

Abbyy FineReader 11 or 12 engine

The predefined profile "BarcodeRecognition_Accuracy" is optimized for this purpose and will detect all supported types of barcodes and QR codes.

Use the profile using the OCR engine running on the OCR Service:

```
using (var engine = Engine.Create("service"))
{
    engine.SetParameters(@"PredefinedProfile=BarcodeRecognition_Accuracy");
    ...
}
```

> **Note:** This profile detects barcodes only and cannot be used to make any text extractable. However, with a profile file barcode and text recognition can be performed in one step

Processing configuration

1. **Activate OCR processing**: Activate OCR processing of all pages:

```
var pageOcr = new PageOcrParams();
pageOcr.Mode = PageOcrMode.All;
```

2. **Enable barcode extraction**: Write all detected barcodes in XML format to a memory stream:

```
var barcodeOcr = new BarcodeParams();
barcodeOcr.Mode = BarcodeMode.Extract;
barcodeOcr.XmlOutput = new MemoryStream();
```

The format of the barcodes XML file is documented in the XML schema `barcodes.xsd` located in the documentation folder of the 3-Heights® PDF OCR API.

Process document

```
var ocr = new OcrParams();
ocr.Engine = engine;

document.Process(output, null, ocr, null,
                 null, pageOcr, barcodeOcr);

barcodes.XmlOutput.Seek(0, SeekOrigin.Begin);
XElement barcodes = XElement.Load(barcodes.XmlOutput, LoadOptions.PreserveWhitespace);

foreach (var barcode in barcodes.Elements(barcodes.Name.Namespace + "barcode"))
{
    ...
}
```

4.3 How to handle conversion warnings

There are two types of problems that may occur when processing a file.

Some problems are severe and hinder further processing. As a result, the processing is aborted and the method `Process` will throw an exception.

Problems that do not hinder further processing are classified as warnings. In this case, the method `Process` will return the list of warnings.

For some processes, certain warnings might be critical, e.g. as described in How to make text extractable. In order to simplify the processing of warnings, they are divided into categories.

The categories are described in `WarningCode`.

Example: This example demonstrates, how the warning's category can be used to classify certain warnings as critical.

```
var warnings = document.Process(output, ...);
foreach (var warning in warnings)
{
    if (warning.Code == WarningCode.Text)
        throw new Exception(warning.Message);
}
```

4.4 How to optimize the performance

Minimize the number of OCR operations

A page is processed by the OCR engine only, if required by either the `ImageOcrMode`, `TextOcrMode`, or `PageOcrMode`. Therefore, these modes should be set carefully, such that no unnecessary OCR operations are performed.

Use one or more OCR Service instances

With the OCR Service, multiple pages can be OCR processed concurrently. For this reason, it is recommended to use the OCR Service plugin.

Parallel processing is controlled by:

- the number of running OCR Services
- the number of parallel processes configured in each OCR Service
- the version of the OCR Service, which should be 4.11.20.0 or higher

Optimize OCR engine performance

By setting apt OCR engine parameters, the performance can be improved. Specifically, it is recommended to deactivate all features that are not required or use a specialized predefined profile. Consult the manual of the OCR engine for more information.

Mass OCR processing

When processing multiple documents, the same `Engine` instance can and should be used.

The 3-Heights® PDF OCR API is fully thread-safe. So multiple documents can be processed concurrently, as longs as each instance of `Engine` and `Document` is used in one thread at the time only.

Note that some OCR engines are not thread-safe. This is one of the reasons, why the use of an OCR Service is recommended.

4.5 Thread safety

The 3-Heights® PDF OCR API is fully thread-safe with one rule:

An object may only be accessed in one thread concurrently.

4.6 Garbage collection and closing objects

Every interface object is considered being a resource that needs to be closed after use. Most objects are closed automatically, at the latest when the owning document is closed, in C# and Java possibly earlier by the garbage collector.

5 Programming interfaces

Where possible and useful, the 3-Heights® PDF OCR API uses language specific features. This means that some parts of the API use different syntax for different programming languages.

5.1 .NET interface

5.1.1 `IDisposable` Objects

Objects that must be closed explicitly implement the `IDisposable` interface. Instead of calling `Dispose()` directly, it is recommended that you use the "`using`" statement:

```
using (Document document = ...)
{
  ...
} // document.Dispose() is called implicitly here
```

See also Garbage collection and closing objects.

5.1.2 Error handling

Errors are reported using exceptions.

The three logic error codes are mapped to the corresponding native exception classes:

`IllegalArgument` maps to `System.ArgumentException`

`IllegalState` maps to `System.InvalidOperationException`

`UnsupportedOperation` maps to `System.NotSupportedException`

The rest of the error codes is modeled using a single exception class `PdfTools.ErrorCodeException` that provides access to the underlying error code and message.

5.1.3 Streams

The native stream interface `System.IO.Stream` is used.

5.1.4 Lists

Lists implement the native list interface `System.Collections.Generic.IList<T>`.

5.2 Java interface

5.2.1 `AutoCloseable` Objects

Objects that must be closed explicitly implement the `AutoCloseable` interface. Instead of calling `close()` directly, it is recommended that you use the "try-with-resources" statement:

```
try (Document document = ...) {
  ...
} // document.close() is called implicitly here
```

See also Garbage collection and closing objects.

5.2.2 Properties

Properties are modeled with setter and getter methods.

5.2.3 Error handling

Errors are reported using exceptions.

The three logic error codes are mapped to the corresponding native runtime exception classes and are not checked:

`IllegalArgument` maps to `java.lang.IllegalArgumentException`

`IllegalState` maps to `java.lang.IllegalStateException`

`UnsupportedOperation` maps to `java.lang.UnsupportedOperationException`

The rest of the error codes is modeled using a single checked exception class `com.pdf_tools.ErrorCodeException` that provides access to the underlying error code and message.

5.2.4 Streams

The native stream interfaces cannot be used, because they are lacking two important features:

- The PDF file format is based on random access. Native Java streams have only limited support for this.
- The ability to read from an output stream is crucial for processing large files.

Instead, a custom stream interface `com.pdf_tools.Stream` is provided, which has a similar interface as `java.io.RandomAccessFile`.

An implementation for files is provided, backed by `java.io.RandomAccessFile`.

5.2.5 Lists

Lists implement the native Java list interface `java.util.List`.

5.3 C interface

5.3.1 Namespaces, classes, and methods

In most languages, namespaces and classes are used to model the interfaces.

The exception is C, where this is modeled with function prefixes and functions operating on handles. The prefix of all functions of the 3-Heights® PDF OCR API is `PdfOcr`, for types, it is `TPdfOcr` and for enum values, `ePdfOcr`.

5.3.2 Library initialization

The first method called must be PdfOcrInitialize. Failing to invoke this function results in undefined behavior. Similarly, the last method must be PdfOcrUninitialize.

5.3.3 Objects

Objects in the C interface are represented by object handles. After use, all object handles returned by the 3-Heights® PDF OCR API must be closed with PdfOcrClose.

5.3.4 Properties

Properties are modeled with setter and getter methods.

5.3.5 Error handling

After a having called a method, an error should be detected as follows:

a. If the method's return type is BOOL or a pointer, and the return value is FALSE or NULL, respectively, then an error has occurred.
b. If the method's return type is other than BOOL or a pointer, then PdfOcrGetLastError must be called to detect whether an error has occurred.

More information about the error can be retrieved by using the functions PdfOcrGetLastError and PdfOcrGetLastErrorMessage.

5.3.6 Strings

All functions involving strings are provided in two different flavors:

- UTF-16 function with suffix W, using WCHAR as parameter type.
- Multibyte character set function with suffix A, using char as parameter type. The concrete character set that is used depends on the platform:
  - On Windows, the current ANSI code page (CP_ACP) is assumed.
  - On Linux or macOS, the current C encoding (LC_CTYPE) is used.

In addition to the effective function names with suffix, there's a macro without suffix for each function pair: It either resolves to the W variant (if _UNICODE is defined), or to the A variant (if _UNICODE is **not** defined).

Example: Signature of an API string property setter, where ‹String› stands for the property's name:

```
void PdfOcrSet‹String›A(const char* szString);  // Multibyte encoding
void PdfOcrSet‹String›W(const WCHAR* szString); // UTF-16
#ifdef _UNICODE
#define PdfOcrSet‹String› PdfOcrSet‹String›W
#else
#define PdfOcrSet‹String› PdfOcrSet‹String›A
#endif
```

String return values

Functions that return a string are treated specially in C. Instead of returning the string, those functions take a buffer and size as last parameters and write into that buffer. The return value is the amount of data written to the buffer.

To determine the required buffer size, the function has to be called with `NULL` as argument.

Calling the function with a buffer size that is too small results in a `ePdfErrorIllegalState`.

Multibyte character set functions (with suffix `A`) that return a string can fail to encode the string in the current operating systems' encoding. In case of such a failure, the return value is `0` and no error code is set. To prevent such failures, it is recommended that you use the UTF-16 (`W`) functions on Windows or to use operating systems with a Unicode code page.

Example: Signature and usage of an API string property getter (error handling is omitted), where `‹String›` stands for the property's name:

```
size_t PdfOcrGet‹String›A(char* pBuffer, size_t nBufferSize);
```

```
size_t nBufferSize = PdfOcrGet‹String›A(NULL, 0);
char* pBuffer = malloc(nBufferSize * sizeof char);
nBufferSize = PdfOcrGet‹String›A(pBuffer, nBufferSize);
```

5.3.7 Streams

Streams are modeled by means of a set of callbacks and a context pointer, grouped in a struct `TPdfStreamDescriptor`.

An implementation for `FILE*` is provided in the header file `pdfdecl.h`. (Search for function `PdfCreateFILEStreamDescriptor`.)

5.3.8 Lists

Lists in C are implemented like any other interface.

`List` Interface

Every list type provides a subset of the following properties and methods, where `‹ElementType›` stands for the type name of the contained elements:

`Count`

Property (get): `int Count`

The number of elements of the list.

Get

Method: `‹ElementType› Get(int index)`
Error codes: `IllegalState`, `IllegalArgument`, `UnsupportedOperation`

Get an element of the list.

Append

Method: `Append(‹ElementType› element)`
Error codes: `IllegalState`, `IllegalArgument`, `UnsupportedOperation`

Append an element to the list.

6 Interface reference

> **Note:** This chapter describes the C# interface of the 3-Heights® PDF OCR API. Other interfaces work similarly, i.e. they have calls with similar names and the call sequence to be used is the same as with C#.
> See Java interface and C interface for more information.

6.1 Common elements

> **Note:** In this section, common static methods and properties are listed. They can be called in every interface.

6.1.1 CheckLicense

Method: `void CheckLicense()`
Static
Error code: `License`

Check if the product is properly licensed.

This method can be used to perform a license check without actually opening or creating a document, e.g. when starting a GUI application or a service.

Error Code:

`License` The product is not properly licensed.

6.1.2 LicenseKey

Property (set): `String LicenseKey`
Static
Error code (set): `License`

Set the license key.

> **Note:** License keys that require activation can only be installed in the license manager. Setting them at runtime is not supported.

Error Code:

`License` The license key is not valid.

6.1.3 ProductVersion

Property (get): `String ProductVersion`
Static

Get the version of the 3-Heights® PDF OCR API in the format “A.C.D.E”.

%

6.2 Engine Interface

Typically when processing documents, an OCR engine is required. They can be created using the method `Create`.

OCR engines can be reused to process multiple files. However, one OCR engine can only be used to process one file at the time.

Note that some OCR engines must be disposed in the same thread where they have been created.

Note that of some OCR engines only one instance can be created per process.

6.2.1 Create

Method: `Engine Create(String name)`
Static
Error code: `Infrastructure`

Create a new OCR engine object.

The list of available OCR engines can be retrieved using `Engines`.

Optionally the `name` argument may be followed by `@` and engine creation parameters, e.g. `"service@http://localhost:7982/"`.

Parameter:

`name` `[String]` The engine name and optional parameters.

Returns:

The newly created engine instance.

Error Code:

`Infrastructure` If the `name` argument is invalid.

6.2.2 Engines

Property (get): `Engine[] Engines`
Static

Get the list the names of all available OCR engines.

6.2.3 SetLanguages

Method: `void SetLanguages(String languages)`
Error code: `Infrastructure`

Set OCR engine specific language settings, e.g. “German,English”. This can be used to improve detection accuracy.

Note that for some engines it is crucial to set the used languages correctly. For example, Abbyy FineReader 11 will only detect characters used in these languages. So if `languages` is set to “English” only, umlauts such as “äöü” are reconigzed as “aou”.

Error Code:

`Infrastructure` If the `languages` argument is invalid.

6.2.4 SetParameters

Method: `void SetParameters(String parameters)`
Error code: `Infrastructure`

Set OCR engine specific parameters.

OCR engine specific parameters (key/value pairs) can be set to optimize the performance or activate optional recognition features.

Error Code:

`Infrastructure` If the `parameters` argument is invalid.

6.2.5 RemainingPageCredits

Property (get): `int RemainingPageCredits`

Get the number of remaining page credits.

This is relevant for OCR engines used with a license that limits the number of pages. In all other cases, a count of 2147483647 or higher will be reported.

6.3 Document Interface

6.3.1 Open

Method: `Document Open(Stream stream, String password)`
Static
Error codes: `Password`, `IO`, `Corrupt`, `Conformance`, `IllegalArgument`

Open a PDF document.

Parameters:

`stream` `[Stream]` The stream where the PDF document is stored.

Random read access is required.

`password` `[String]` The password to open the PDF document.

Returns:

The newly created document instance.

Error Codes:

`Password` The file is encrypted and the `password` is not valid.

`IO` Error reading from the stream.

`Corrupt` The file is corrupt or not a PDF document.

`Conformance` The document's conformance is not supported.

`IllegalArgument` If the `stream` argument is `null`.

6.3.2 Process

Method: `Warning[] Process(Stream stream, EncryptionParams encryption, OcrParams ocr, ImageOcrParams imageOcr, TextOcrParams textOcr, PageOcrParams pageOcr, BarcodeParams barcodes)`
Error codes: `IO`, `IO`, `Conformance`, `Infrastructure`, `Processing`, `IllegalState`, `IllegalArgument`

Process the document.

See chapter Process description for more information on how documents are processed.

Parameters:

stream [Stream] The stream to which the output is written. stream must support both random read and write access.

encryption [EncryptionParams] Optional encryption parameters.

ocr [OcrParams] Optional OCR parameters.

imageOcr [ImageOcrParams] Optional image OCR parameters.

textOcr [TextOcrParams] Optional text OCR parameters.

pageOcr [PageOcrParams] Optional page OCR parameters.

barcodes [BarcodeParams] Optional parameter specifying how recognized barcodes should be processed.

This parameter has only an effect, if pages containing barcodes are processed and detected by the OCR engine. This depends on the type and settings of the engine in ocr as well as the parameters imageOcr, textOcr, and pageOcr.

Returns:

A list of Warning objects that occurred during processing of the document.

Error Codes:

IO Parameter XmlOutput required but not valid.

Conformance If encryption parameters are set for a PDF/A file.

Infrastructure If the OCR engine is not operational, e.g. due to a license issue or TCP connection problems.

Processing If the document cannot be processed.

IllegalState If the document has already been closed.

IllegalArgument If an OCR engine is required but no valid one is specified.

6.4 Warning Interface

A warning object describes an event that occurred in Process. Warnings are non-critical in a way, that the operation must not be aborted by the 3-Heights® PDF OCR API. Nonetheless, it is recommended to review warnings that occur and decide, which must be treated as critical errors for a particular process.

6.4.1 Code

Property (get): WarningCode Code

The code describing the type of the warning.

6.4.2 Message

Property (get): `String Message`

The message describing the warning.

6.4.3 PageNo

Property (get): `int PageNo`

The number of the page on which the warning occurred.

6.5 Structures

6.5.1 BarcodeParams Struct

See chapter How to detect barcodes for an example of how to use barcode parameters.

`Mode` [`BarcodeMode`] The mode according to which barcodes are processed.

default: `BarcodeMode.None`

`XmlOutput` [`Stream`] The stream to which recognized barcodes are written. The format of the resulting barcodes XML file is documented in the XML schema `barcodes.xsd` located in the documentation folder of the 3-Heights® PDF OCR API.

This property is required, if and only if the property `Mode` is `BarcodeMode.Extract`.

default: `null`

6.5.2 EncryptionParams Struct

`OwnerPassword` [`string`] The owner password of the document.

This password is also referred to as the author's password and grants full access to the document. Not only can the document be opened and read, it also allows you to change the document's security settings (access permission and passwords).

`UserPassword` [`string`] The user password of the document.

Protects the document against unauthorized opening and reading. If a PDF document is protected by a user password, either the user or owner password must be provided to open and read the document. If a document has a user password, it must have an owner password as well. If no owner password is defined, the owner password is the same as the user password.

`UserPermissions` [`Permission`] The user permissions.

The operations in a PDF document to be granted is controlled via these permission flags. To set permission flags, the PDF document must be encrypted and have an owner password. The owner password is required to initially set or later change the permission flags.

6.5.3 ImageOcrParams Struct

The image OCR parameters control under what conditions and how images should be processed.

> **Note:** The properties `RotateScan` and `DeskewScan` have an effect only, if:
> 1. The page is a scan and not born-digital.
> 2. The page is processed by the OCR engine, which depends on the `ImageOcrMode` set.
> 3. The required information is provided by the OCR engine, which depends on the type and settings of the engine.

`RotateScan` [`bool`] Whether scanned pages should be rotated according to the orientation detected by the OCR engine.

default: `false`

`DeskewScan` [`bool`] Whether scanned pages should be deskewed according to the angle detected by the OCR engine.

default: `false`

`Mode` [`ImageOcrMode`] The mode according to which images are processed.

See chapter Process description for a description on how setting this property affects OCR processing.

default: `ImageOcrMode.None`

`RemoveOnlyTr3Text` [`bool`] Whether to remove only ocr text using text rendering mode 3. Ths option has an effect when using `ImageOcrMode.ReplaceText` or `ImageOcrMode.RemoveText`.

default: `false`

6.5.4 OcrParams Struct

`Engine` [`Engine`] The OCR engine (see `Engine`).

This parameter may not be `null` for most combinations of `ImageOcrMode`, `TextOcrMode`, and `PageOcrMode`.

default: `null`

`OcrDPI` [`double`] The default resolution in DPI used for OCR.

Each page's optimal OCR resolution is determined automatically, such that all images and text can be recognized. The default resolution is chosen, if it is within the range of optimal resolutions. The range of allowed resolutions can be chosen using `OcrMinDPI` and `OcrMaxDPI`.

The range should be in the range of resolutions supported by the OCR engine. Most OCR engines are optimized for resolutions around 300 DPI. Selecting a resolution that is too low will hinder the detection of small text. An excessively high resolution will reduce performance because of the higher resource requirements to render the page images and perform OCR on them.

If the optimal resolution of a page is not within the range, an OCR warning is generated.

default: `300`

`OcrMinDPI` [`double`] The minimum resolution in DPI used for OCR.

default: `200`

`OcrMaxDPI` [`double`] The maximum resolution in DPI used for OCR.

default: `400`

`ProcessEmbeddedFiles` [`bool`] Whether embedded files should be processed recursively. Otherwise embedded files are copied as-is.

default: `false`

6.5.5 PageOcrParams Struct

`Mode` [`PageOcrMode`] The mode according to which pages are processed.

See chapter Process description for a description on how setting this property affects OCR processing.

default: `PageOcrMode.None`

`Tagging` [`TaggingMode`] "Tagging" adds structural information to a PDF. This information can be used e.g. to read the document to the visually impaired.

This property controls, if this detected "tagging" information is generated for OCR text.

default: `TaggingMode.Auto`

6.5.6 TextOcrParams Struct

`Mode` [`TextOcrMode`] The mode according to which text is processed.

See chapter Process description for a description on how setting this property affects OCR processing.

default: `TextOcrMode.None`

`TextOcrSkip` [`TextOcrSkip`] Defines text that can be skipped from text OCR processing

default: `TextOcrSkip.None`

`ToUnicodeSource` [`ToUnicodeSource`] Defines additional ToUnicode sources, i.e. in addition to OCR processing.

default: `ToUnicodeSource.None`

6.6 Enumerations

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

6.6.1 BarcodeMode Enumeration

`None` Do not add barcode information.

`Embed` Embedded recognized barcodes into the document's XMP metadata.

Extract Write recognized barcodes to the stream specified by XmlOutput.

6.6.2 ImageOcrMode Enumeration

None Do not process images.

UpdateText Only process images that have no OCR text.

ReplaceText Process all images and remove existing OCR text.

RemoveText Remove existing OCR text.

IfNoText Process images only if document contains no text.

6.6.3 PageOcrMode Enumeration

None Do not process pages.

All Process all pages that are not empty.

IfNoText Process all pages that contain content but no text.

AddResults Do not trigger processing of pages. But if pages are OCR processed, e.g. due to another OCR mode, add results as OCR text to pages.

6.6.4 Permission Enumeration

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

Print Low resolution printing

Modify Changing the document

Copy Content copying or extraction

Annotate Annotations

FillForms Filling of form fields

SupportDisabilities Support for disabilities

Assemble Document assembly

DigitalPrint High resolution printing

6.6.5 TaggingMode Enumeration

None Do not add tagging information.

Update Force embedding of tagging information. A warning is generated, if no tagging information can be added. Therefore, this value is recommended if tagging information is crucial to your process.

Auto Determine tagging mode automatically. Use Update for scans and born-digital documents with tagging, and None otherwise.

6.6.6 TextOcrMode Enumeration

None Do not process text.

Update Only process text that is not extractable.

For all characters that have no meaningful Unicode, OCR processing is used to determine the Unicode. This is the recommended mode to make text extractable.

Note that making text extractable requires many OCR operations. The reason is that of all characters multiple instances must be recognized, to deal with erroneous OCR recognitions.

`Replace` Process all text.

OCR is used to determine the Unicode of all characters, that is even if they seemingly have Unicode information. This is useful for documents that possibly contain wrong Unicode information. Wrong Unicode information is typically created by flawed PDF creators or to obfuscate text (i.e. to prevent copy-and-paste or search operations).

For documents that contain correct Unicode information, this mode produces the same result as the mode Update. The rare exceptions are special fonts for which the OCR engine produces wrong results, which might happen for some decorative or handwritten fonts. The main disadvantage of the mode Replace over Update is, that more OCR operations are required.

6.6.7 `TextOcrSkip` Enumeration

`None` Do not skip any text in text ocr processing.

`KnownSymbolic` Skip text of all fonts that are known to be symbolic, e.g. "ZapfDingbats" or "Wingdings".

For many symbols of these fonts there exist no Unicodes. Also, even the ones that have Unicodes, such as "✓" or "→", cannot be recognized by most OCR engines. Therefore, OCR processing of these fonts usually does not produce a meaningful result and could be skipped.

`Pua` Skip text with Unicodes from Private Use Areas (PUA), i.e. accept Unicodes from PUA as meaningful.

Unicodes from the PUA are typically used for symbols, for which no Unicodes exist. OCR processing of these symbols does not produce a result and could be skipped.

On the other hand, some bad PDF creators use PUA for normal text. For these cases, OCR processing should be performed.

6.6.8 `ToUnicodeSource` Enumeration

`None` Do not use any additional sources. Only use ToUnicode information contained in the PDF document as described in the PDF Reference.

`KnownSymbolicPua` Use Unicodes from Private Use Areas (PUA) for all fonts that are known to be symbolic, e.g. "ZapfDingbats" or "Wingdings".

`FallbackAllPua` Use Unicodes from Private Use Areas (PUA) for all characters for which no better Unicode can be determined.

`InstalledFont` If on the system a font of the same name is installed, use Unicodes of matching glyphs.

6.6.9 `WarningCode` Enumeration

See chapter How to handle conversion warnings for more information on how to use `WarningCode`.

`Ocr` The warning is related to OCR recognition.

`Tagging` The warning is related to tagging. It must be considered critical when tagging documents for accessibility as described in How to tag scans for accessibility (PDF/A level A). Note that this warning does not mean, that OCR text has not been added, but merely that there was an issue with tagging it.

Text The warning is related to making text extractable. This is critical when making text extractable as described in How to make text extractable.

Signed Processing a signed file changes it, such that all signatures become invalid. Therefore, all signatures are removed and this warning is generated.

6.6.10 ErrorCode Enumeration

See Programming interfaces for more information about how these codes are mapped to exceptions in the .NET and Java interface.

Logic errors

These codes denote errors in the application program logic and should never happen at runtime.

UnsupportedOperation The requested method or property is not supported.

IllegalState The object is in a state where the requested object or property cannot be called.

IllegalArgument The method was called using an illegal argument.

Environmental errors

Generic The error is not further specified.

Fatal A fatal error occurred.

License Licensing error.

NotFound The requested item or resource could not be found.

IO Error while reading or writing from a stream.

UnknownFormat The format is unknown.

Corrupt The data is corrupt.

Password The resource or document is protected by a password.

Conformance A conformance mismatch happened.

UnsupportedFeature The file contains an unsupported feature.

Infrastructure An infrastructure error occurred.

Processing The file cannot be processed.

Exists The item already exists.

7 Version history

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

7.1 Changes in versions 6.19–6.27

- **Update** license agreement to version 2.9

7.2 Changes in versions 6.13–6.18

No functional changes.

7.3 Changes in versions 6.1–6.12

- **New** version of OCR plugin `"barcodes"` with QR Code recognition improvements.
- [.NET] **Improved**`ErrorCodeException` which is now serializable (except for the .NET Standard 1.0 target).
- **Improved** search algorithm for installed fonts: User fonts under Windows are now also taken into account.
- [Java] **Changed** minimal supported Java language version to 7 [previously 6].
- [.NET] **New** availability of this product as NuGet package for Windows and Linux.
- [.NET] **New** support for .NET Core versions 1.0 and higher. The support is restricted to a subset of the operating systems supported by .NET Core, see Operating systems.
- [.NET] **Changed** platform support for NuGet packages: The platform "AnyCPU" is now supported for .NET Framework projects.
- [Java] **New** static method `fromValue` for all non-flags enums.

Interface `Document`

- [Java] **Changed** inheritance. Interface now inherits from `AutoCloseable` and therefore can be used in a try-resource clause.

Interface `EngineList`

- [Java] **Deprecated** method `close`.
- [.NET] **Deprecated** method `Dispose`.

Interface `WarningList`

- [Java] **Deprecated** method `close`.
- [.NET] **Deprecated** method `Dispose`.

Interface `Engine`

- [Java] **Deprecated** method `close`.

- [.NET] **Deprecated** method `Dispose`.

Interface `Warning`

- [Java] **Deprecated** method `close`.
- [.NET] **Deprecated** method `Dispose`.

Structure `TextOcrParams`

- **New** property `TextOcrSkip` to define text that can be skipped from text OCR processing.
- **New** property `ToUnicodeSource` to enable additional sources of ToUnicode information, i.e. in addition to OCR processing.

Structure `PageOcrParams`

- **Changed** default value of property `Tagging` to new value `TaggingMode.Auto`.

Enummeration `TaggingMode`

- **New** value `TaggingMode.Auto` to enable automatic detection whether tagging information should be added or not.

7.4 Changes in version 5

- **Improved** wording of warning messages. The new messages do not contain the words “warning” nor “error”, such that they can be used as both warning and error message.
- **New** warning, if signatures were removed.
- **New** version of OCR plugin `"barcodes"` with QR Code recognition improvements.
- **New** additional supported operating system: Windows Server 2019.
- **Improved** extraction of recognized barcodes to extract their type.
- [.NET] **Changed**`PdfOcrNET.dll` library. Cross-product functionality is outsourced into common library `PdfCommonNET.dll`.

7.5 Changes in version 4.12

- **Introduced** license features `Service`, `Ocr`, and `Barcode`.
- **Improved** image ocr mode to cache OCR text of images. Images that occur on multiple pages must be OCR processed once only.
- **New** specialized OCR plugin `"barcodes"` to recognize barcodes and QR codes without an additional OCR engine.
- **New** OCR plugin `"abbyy12"` for the ABBYY FineReader 12 engine.
- **Improved** memory consumption of product.
- **New** detecion of OCR text that is under images.
- **Improved** ocr result processing, notably the accuracy of associating OCR text to existing text on page.
- **New** HTTP proxy setting in the GUI license manager.
- **New** `PageOcrMode` `AddResult`.

8 Licensing, copyright, and contact

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

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

Contact

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