3-Heights™
PDF Analysis & Repair API

Version 6.12.0
6 Interface Reference ................................................................. 25
6.1 PDFRepair Interface ............................................................. 25
6.1.1 AnalysisOptions .............................................................. 25
6.1.2 Analyze ............................................................................ 25
6.1.3 AnalyzeAndRepair ............................................................ 25
6.1.4 Close ............................................................................... 26
6.1.5 Diagnosis ......................................................................... 26
6.1.6 ErrorCode ......................................................................... 26
6.1.7 ErrorLevel ......................................................................... 26
6.1.8 ErrorMessage .................................................................... 26
6.1.9 GetFirstError .................................................................... 27
6.1.10 GetNextError .................................................................... 27
6.1.11 GetPdf ............................................................................... 27
6.1.12 LicenseIsValid ................................................................. 28
6.1.13 Open ............................................................................... 28
6.1.14 OpenMem ......................................................................... 28
6.1.15 ProductVersion ................................................................. 29
6.1.16 RebuildOptions ............................................................... 29
6.1.17 RecoveryOptions ............................................................. 29
6.1.18 Repair ............................................................................... 29
6.1.19 ReportingLevel ............................................................... 30
6.1.20 SaveAs ............................................................................ 30
6.1.21 SaveInMemory ................................................................. 31
6.1.22 SetLicenseKey .................................................................. 31
6.2 PDFError Interface ............................................................... 31
6.2.1 Count ............................................................................... 31
6.2.2 ErrorCode ......................................................................... 31
6.2.3 Message ............................................................................ 32
6.2.4 ObjectNo ........................................................................... 32
6.2.5 PageNo ............................................................................. 32
6.3 Enumerations ....................................................................... 32
6.3.1 TPDFAnalysisOptionEnumeration .................................. 32
6.3.2 TPDFDiagnosisEnumeration ........................................... 32
6.3.3 TPDFErrorCodeEnumeration ........................................ 33
6.3.4 TPDFPermissionEnumeration .......................................... 33
6.3.5 TPDFRebuildOptionEnumeration .................................... 34
6.3.6 TPDFRecoveryOptionEnumeration .................................. 34

7 Version History ................................................................. 35
7.1 Changes in Version 6 .......................................................... 35
7.2 Changes in Version 5 .......................................................... 35
7.3 Changes in Version 4.12 ...................................................... 35
7.4 Changes in Version 4.11 ...................................................... 35
7.5 Changes in Version 4.10 ...................................................... 35
7.6 Changes in Version 4.9 ....................................................... 36
7.7 Changes in Version 4.8 ....................................................... 36

8 Licensing, Copyright, and Contact ............................................. 37
1 Introduction

1.1 Description

The 3-Heights™ PDF Analysis & Repair API tool is used to analyze, repair and restore the content of corrupt PDF documents.

Unfortunately, the number of corrupt PDF documents is incredibly huge. The cause is usually down to defective generating tools, converters and other influences such as attempts at manual editing, copying via FTP without correct settings, system crashes during PDF creation, network interruptions, defective copying on optical media, etc.

The result leads to an enormous loss of important information and to production downtimes caused by corrupt PDF documents.

The 3-Heights™ PDF Analysis & Repair API analyzes PDF documents with regard to PDF specifications. Defective files are automatically repaired as far as possible and unreadable data is restored.

![PDF Analysis & Repair Tool Diagram]

1.2 Functions

PDF Analysis & Repair API is used to check and, where indicated, repair PDF documents. Users can determine customized profiles from a broad range of analysis and repair options. An exact and detailed description is issued for each reported error. The tool is also capable of reading and processing encrypted PDF files without any problems.

1.2.1 Features

- Analyze and/or repair one or more PDF Documents
- Set analysis options, including:
1.2.2 Formats

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

Output Formats
- PDF 1.x (PDF 1.0, ..., PDF 1.7)
- PDF 2.0

1.2.3 Conformance

Standards:
- ISO 32000-1 (PDF 1.7)
- ISO 32000-2 (PDF 2.0)

1.3 Interfaces

The following interfaces are available:
- C
- Java
1.4 Operating Systems

The 3-Heights™ PDF Analysis & Repair API is available for the following operating systems:

- Windows Client 7+ | 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
- macOS 10.10+ | x64

‘+’ indicates the minimum supported version.

1.5 How to Best Read this Manual

If you are reading this manual for the first time, i.e. would like to evaluate the software, the following steps are suggested.

1. Read the chapter **Introduction** to verify this product meets your requirements.
2. Identify what interface your programming language uses.
3. Read and follow the instructions in the chapter **Installation and Deployment**.
4. In the chapter **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. For a start it is generally best to refer to these samples rather than writing code from scratch.
5. (Optional) Read the chapter **User’s Manual** for general information about the API. Read the **Interface Reference** for specific information about the functions of the API.
2 Installation and Deployment

2.1 Windows

The 3-Heights™ PDF Analysis & Repair API comes as a ZIP archive or as a NuGet package.

The installation of the software requires the following steps.

1. You need administrator rights to install this software.
2. Log in to your download account at [http://www.pdf-tools.com](http://www.pdf-tools.com). Select the product “PDF Analysis & Repair API”. If you have no active downloads available or cannot log in, please contact pdfsales@pdf-tools.com for assistance.
   
   You will find different versions of the product available. We suggest to download the version, which is selected by default. A different version can be selected using the combo box.
   
   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, do the following. Unzip the archive to a local folder, e.g. C:\Program Files\PDF Tools AG\.
   
   This creates the following subdirectories (see also Zip Archive):

<table>
<thead>
<tr>
<th>Subdirectory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>Contains the runtime executable binaries.</td>
</tr>
<tr>
<td>doc</td>
<td>Contains documentation.</td>
</tr>
<tr>
<td>include</td>
<td>Contains header files to include in your C/C++ project.</td>
</tr>
<tr>
<td>jar</td>
<td>Contains Java archive files for Java components.</td>
</tr>
<tr>
<td>lib</td>
<td>Contains the object file library to include in your C/C++ project.</td>
</tr>
<tr>
<td>samples</td>
<td>Contains sample programs in various programming languages</td>
</tr>
</tbody>
</table>

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 which interface you are using. Perform the specific installation steps for that interface described in Interface Specific Installation Steps.

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 Analysis & Repair API provides two interfaces:

- Java interface
- Native C interface

Here is an overview of the files that come with the 3-Heights™ PDF Analysis & Repair API:
### File Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin/x64/libPdfRepairAPI.so</td>
<td>This is the shared library that contains the main functionality. The file’s</td>
</tr>
<tr>
<td></td>
<td>extension differs on macOS (.dylib instead of .so).</td>
</tr>
<tr>
<td>doc/<em>.</em></td>
<td>Documentation</td>
</tr>
<tr>
<td>include/*.h</td>
<td>Contains header files to include in your C/C++ project.</td>
</tr>
<tr>
<td>jar/REPA.jar</td>
<td>Java API archive.</td>
</tr>
<tr>
<td>samples</td>
<td>Example code.</td>
</tr>
</tbody>
</table>

#### 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:

   ```bash
   ldd libPdfRepairAPI.so
   ```

   In case the above reports any missing libraries you have three 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.
   c. Use GNU shared libraries provided by PDF Tools AG:
      2. Download the GNU shared libraries for your platform.
      3. Install the libraries manually according your system's documentation. This typically involves copying them to your library directory, e.g. /usr/lib or /usr/lib64, and running ldconfig.
      4. Verify that the GNU shared libraries required by the product are available on your system now.
3. Create a link to the shared library from one of the standard library directories, e.g:

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

4. Optionally register your license key using the license manager.
5. Identify which interface you are using. Perform the specific installation steps for that interface described in Interface Specific Installation Steps.

#### 2.2.2 macOS

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

```bash
ln libPdfRepairAPI.dylib libPdfRepairAPI.jnilib
```
2.3 Zip Archive

The 3-Heights™ PDF Analysis & Repair API provides four different interfaces. The installation and deployment of the software depend on the interface you are using. The table below shows the supported interfaces and examples with which programming languages they can be used.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Programming Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NET</td>
<td>The MS software platform .NET can be used with any .NET capable programming language such as:</td>
</tr>
<tr>
<td></td>
<td>- C#</td>
</tr>
<tr>
<td></td>
<td>- VB .NET</td>
</tr>
<tr>
<td></td>
<td>- J#</td>
</tr>
<tr>
<td></td>
<td>- others</td>
</tr>
<tr>
<td></td>
<td>For a convenient way to use this interface, see NuGet Package.</td>
</tr>
<tr>
<td>Java</td>
<td>The Java interface is available on all platforms.</td>
</tr>
<tr>
<td>COM</td>
<td>The component object model (COM) interface can be used with any COM-capable programming language, such as:</td>
</tr>
<tr>
<td></td>
<td>- MS Visual Basic</td>
</tr>
<tr>
<td></td>
<td>- MS Office Products such as Access or Excel (VBA)</td>
</tr>
<tr>
<td></td>
<td>- C++</td>
</tr>
<tr>
<td></td>
<td>- VBScript</td>
</tr>
<tr>
<td></td>
<td>- others</td>
</tr>
<tr>
<td></td>
<td>This interface is available in the Windows version only.</td>
</tr>
<tr>
<td>C</td>
<td>The native C interface is for use with C and C++. This interface is available on all platforms.</td>
</tr>
</tbody>
</table>

2.3.1 Development

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

- **Req.**  This file is required for this interface.
- **Opt.**  This file is optional. See also table *File Description* to identify which files are required for your application.
- **Doc.**  This file is for documentation only.
- **Empty field**  An empty field indicates this file is not used at all for this particular interface.

<table>
<thead>
<tr>
<th>Files for Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>bin*platform\PdfRepairAPI.dll</td>
</tr>
<tr>
<td>bin*NET.dll</td>
</tr>
<tr>
<td>bin*NET.xml</td>
</tr>
</tbody>
</table>
The purpose of the most important distributed files of is described in table **File Description**.

<table>
<thead>
<tr>
<th>Name</th>
<th>.NET</th>
<th>Java</th>
<th>COM</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc\PdfRepairAPI.idl</td>
<td>Doc.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc\javadoc*.*</td>
<td>Doc.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>include\pdfrepairapi_c.h</td>
<td>Req.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>include*.*</td>
<td>Opt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jar\REPA.jar</td>
<td>Req.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib&lt;platform&gt;\PdfRepairAPI.lib</td>
<td>Req.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**File Description**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin&lt;platform&gt;\PdfRepairAPI.dll</td>
<td>This is the DLL that contains the main functionality (required), where &lt;platform&gt; is either Win32 or x64 for the 23-bit or the 64-bit library respectively.</td>
</tr>
<tr>
<td>bin*NET.dll</td>
<td>The .NET assemblies are required when using the .NET interface. The files bin*NET.xml contain the corresponding XML documentation for MS Visual Studio.</td>
</tr>
<tr>
<td>doc*.*</td>
<td>Various documentations.</td>
</tr>
<tr>
<td>include*.*</td>
<td>Contains files to include in your C / C++ project.</td>
</tr>
<tr>
<td>lib&lt;platform&gt;\PdfRepairAPI.lib</td>
<td>On Windows operating systems, the object file library needs to be linked to the C/C++ project.</td>
</tr>
<tr>
<td>jar\REPA.jar</td>
<td>The Java API archive.</td>
</tr>
<tr>
<td>samples*.*</td>
<td>Contains sample programs in different programming languages.</td>
</tr>
</tbody>
</table>

**2.3.2 Deployment**

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

---

2 Not required for Linux or macOS.
3 These files must reside in the same directory as PdfRepairAPI.dll.
The deployment of an application works as described below:

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

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

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

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

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

   ```
   regsvr32 /s %targetlocation%PdfRepairAPI.dll.
   ```

### 2.4 NuGet Package

Nuget is a package manager that facilitates the integration of libraries for the software development in .NET. The nuget package for the 3-Heights™ PDF Analysis & Repair API contains all the libraries needed, managed and native.

**Installation**  Download the package PdfTools.PdfRepair.6.12.0.nupkg from your account on https://www.pdf-tools.com/ to some suitable location.

In Visual Studio click on “Tools” and then “Options”. Select “NuGet Package Manager” and add the location of the downloaded package in “Package Sources”.

Right-click on a .NET project in Visual Studio and select “Manage NuGet Packages...”. Finally, select the package source that was defined above and browse to the desired package.


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

In order to use the software, you must first install a license key for the 3-Heights™ PDF Analysis & Repair API. To do this you have to download the product kit and use the license manager in it. See also License Management.
2.5 Interface Specific Installation Steps

2.5.1 COM Interface

**Registration** Before you can use the 3-Heights™ PDF Analysis & Repair API component in your COM application program you have to register the component using the `regsvr32.exe` program that is provided with the Windows operating system. The following command shows the registration of `PdfRepairAPI.dll`. Note that in Windows Vista and later, the command needs to be executed from an administrator shell.

```
regsvr32 "C:\Program Files\PDF Tools AG\bin\<platform>\PdfRepairAPI.dll"
```

Where `<platform>` is `Win32` for the 32-bit and `x64` for the 64-bit version.

If you are using a 64-bit operating system and would like to register the 32-bit version of the 3-Heights™ PDF Analysis & Repair API, you need to use the `regsvr32` from the directory `%SystemRoot%\SysWOW64` instead of `%SystemRoot%\System32`.4

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

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

2.5.2 Java Interface

The 3-Heights™ PDF Analysis & Repair API requires Java version 7 or higher.

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

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

**For execution** Additionally the library `PdfRepairAPI.dll` needs be in one of the system's library directories5 or added to the Java system property `java.library.path`. This can be achieved by either adding it dynamically at program startup before using the API, or by specifying it using the switch `-Djava.library.path` when starting the Java VM. Choose the correct subdirectory (`x64` or `Win32` on Windows) depending on the platform of the Java VM.

---

4 Otherwise you get the following message: `LoadLibrary("PdfRepairAPI.dll")` failed - The specified module could not be found.
5 On Windows defined by the environment variable PATH and e.g. on Linux defined by `LD_LIBRARY_PATH`.
6 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"
Note that on Linux or macOS, the path separator usually is a colon and hence the above changes to something like:

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

### 2.5.3 .NET Interface

The 3-Heights™ PDF Analysis & Repair 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 to 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 to be added as references to the project; They are needed at compile time. PdfRepairAPI.dll is not a .NET assembly, but a native library. It is not to be 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 PdfRepairAPI.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 Pathwin-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”, the above can be achieved by adding the 32-bit or 64-bit PdfRepairAPI.dll respectively as an “existing item” to the project, and setting its property “Copy to output directory” to true. When using the “AnyCPU” platform, then you have to make sure by some other means that both the 32-bit and the 64-bit PdfRepairAPI.dll are copied to subdirectories win-x86 and win-x64 of the output directory respectively.

### 2.5.4 C Interface

- The header file pdfrepairapi_c.h needs to be included in the C/C++ program.
- On Windows operating systems, the library PdfRepairAPI.lib needs to be linked to the project.
- The dynamic link library PdfRepairAPI.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: In order to uninstall the product, undo all the steps done during installation, e.g. un-register using regsvr32.exe /u, delete all files, etc.

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

### 2.7 Note about the Evaluation License

With the evaluation license the 3-Heights™ PDF Analysis & Repair API automatically adds a watermark to the output files.
3 License Management

The 3-Heights™ PDF Analysis & Repair 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 will fail with an error code and error message indicating the reason.

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

4.1 Visual Basic 6

After installing the 3-Heights™ PDF Analysis & Repair API and registering the COM interface (see chapter [COM Interface](#)), you find a Visual Basic example repair.vbp in the directory samples\VB\. You can either use this sample as a base for an application, or you can start from scratch.

The 3-Heights™ PDF Analysis & Repair API is very easy to use.

**Example:** A Visual Basic 6 sample looks as simple as this

```vbnet
Private Sub repair_Click()
    Dim repair As New PDFREPAIRAPILib.PDFRepair
End Sub
```

If a PDF document cannot be repaired and contains images, it is possible to recover the images by setting the property:

```vbnet
repair.RecoveryOptions = repair.RecoveryOptions or eRecoverImages
```

4.2 ASP VBScript

The class name to be used is "PDFREPAIRAPI.PDFRepair".

**Simplified Example:**

```vbs
%@Language=VBScript %>
<%
    option explicit
    dim repair
    set repair = Server.CreateObject("PDFREPAIRAPI.PDFRepair")
    repair.Open("path\file_to_be_repaired.pdf")
    repair.AnalyzeAndRepair()
    repair.SaveAs("path\output_file.pdf")
    repair.Close
%>
```

4.3 .NET

There should be at least one .NET sample for MS Visual Studio available in the ZIP archive of the Windows version of the 3-Heights™ PDF Analysis & Repair API. The easiest for a quick start is to refer to this sample.

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

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

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

4. Write your code.

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

4.3.1 Visual Basic

3. Double-click “My Project” to view its properties. On the left hand side, select the menu “References”. The .NET assemblies you added before should show up in the upper window. In the lower window import the namespaces Pdftools.Pdf, and Pdftools.PdfRepair.

You should now have settings similar as in the screenshot below:
4. The .NET interface can now be used as shown below:

**Example:**

```csharp
Dim doc As New Pdftools.PdfRepair.Repair
' Or if the namespace Pdftools.PdfRepair is imported:
' Dim doc As New Repair
doc.Open(...)```

### 4.3.2 C#

3. Add the following namespaces:

**Example:**

```csharp
using Pdftools.Pdf;
using Pdftools.PdfRepair;
```

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

```csharp
using (Repair doc = new Repair())
{
    doc.Open(...)
    ...
}
```

### 4.3.3 Deployment

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

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

### 4.3.4 Troubleshooting: `TypeInitializationException`

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

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

- **System.DllNotFoundException**   Unable to load DLL PdfRepairAPI.dll: The specified module could not be found.
- **System.BadImageFormatException** An attempt was made to load a program with an incorrect format.

The following sections describe in more detail, how to resolve the respective issue.

#### Troubleshooting: `DllNotFoundException`

This means, that the native DLL PdfRepairAPI.dll could not be found at execution time. Resolve this by either:

- using the [NuGet Package](#).
- adding PdfRepairAPI.dll as an existing item to your project and set its property "Copy to output directory" to "Copy if newer", or
- adding the directory where PdfRepairAPI.dll resides to the environment variable `%Path%`, or
- manually copying PdfRepairAPI.dll to the output directory of your project.
Troubleshooting: BadImageFormatException

The exception means, that the native DLL PdfRepairAPI.dll has the wrong "bitness" (i.e. platform 32 vs. 64 bit). There are two versions of PdfRepairAPI.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 3 possibilities:

**AnyCPU**  This means, that the application will run as a 32-bit process on 32-bit Windows and as 64-bit process on 64-bit Windows. When using AnyCPU, then a different native DLL has to be used, depending on the Windows platform. This can be ensured either when installing the application by installing the matching native DLL, or at application start-up by determining the application's platform and ensuring the matching native DLL is loaded. 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 will always run 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 will always run as 64-bit process. As a consequence the application will not run on a 32-bit Windows system.
5 User’s Manual

5.1 Overview of the API

5.1.1 What is the 3-Heights™ PDF Analysis & Repair API about?

The API provides two main functionalities as its name indicates:
1. Analysis of PDF documents, detect and report corruptions
2. Repair or recover the detected corruptions and save the result in a new PDF document

5.2 How does the API work?

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

1. In a first step the input-document is opened for reading. This is done using the function `Open`.
2. In the next step analysis options are set using the property `AnalysisOptions`. The more analysis options are set, the deeper the analysis goes and the longer it takes.
3. In case the document is going to be repaired later on, the recovery options can be set at this point using the property `RecoveryOptions`. Recovery options can also be set after the analysis and before the repair step. However as we will see, these two steps can be combined, therefore it make sense to set them previously.
4. The document is being analyzed using either the function `Analyze` or `AnalyzeAndRepair`. As a result of the analysis, the document is qualified valid or corrupt.
5. For corrupt documents, all corruption can be listed in an error report. In the case of an analysis-only process, the input-document can now be closed and the process is done.
6. If the process is also to repair the document, it is saved as a new PDF document using the `SaveAs` function. If the user chooses to repair the PDF document, a new PDF document is created. This document is referred to as output-document. The output-document is completely rebuilt from scratch using all readable information from the input-document. This means if a valid PDF document is repaired, a new document is created as well. If a PDF document is corrupt, the PDF can either be repaired or recovered depending on the level of corruption. The difference between repairing and recovering is described in a separate chapter.
7. The input-document is closed using `Close`.

See also Section Concepts.
5.3 Corrupt PDF Documents

5.3.1 How do PDF Documents get corrupted?

**Reason 1: Incorrect PDF creators**  
One needs to be aware that PDF is a complex format, its specification is more than 1000 pages. Within PDF there are embedded objects, such as different types of fonts, images or compressions, which again are complex on their own and have specifications that are even larger than the PDF specification itself.

There are uncountable different PDF products available, and virtually none of them is capable to support everything PDF offers. And only few of them create actually valid PDF. Most freeware, or home-made PDF creators have flaws. These flaws are often not detected initially simply because the widely used PDF viewer applications detect and repair these errors on the fly. The creator of the PDF doesn’t even notice his PDF is corrupt, because the PDF viewer application fixes or ignores the problem silently. A creator often does not have the goal to create a valid PDF, but just a PDF which can be viewed.

**Reason 2: Binary file is damaged**  
PDF is a binary format. Most of its content is compressed. Editing a PDF file with a text editor, or transmitting a PDF in text mode instead of binary mode (e.g. FTP) corrupts the PDF. Partially transmitting a PDF file cuts off part of the document, this loss of information is not recoverable.

There are further reasons, but the two reasons mentioned are certainly the most common.

5.3.2 How to detect Corruptions?

The most obvious way to detect a problem with a PDF document is if it doesn’t open a PDF viewer application, or there is an error message when opening the document, or part of the document cannot be displayed correctly. For most user this is the only situation where they actually are aware the document is corrupt. Any other corruption that has no direct impact to viewing the document is often ignored.

If documents are being archived or must be of good quality for other reasons, they can be analyzed using a PDF analysis tool.

The 3-Heights™ PDF Analysis & Repair API analyzes documents and detects whether they are valid or not according to the PDF specification.

A simpler test to see whether a document is valid or not is to open it in Adobe Acrobat Professional and close it again. If one is prompted to save the document, it can be an indication that the document was corrupt and was repaired and the repaired document is now displayed to the user. This test however does not provide any information about what was corrupt, i.e. what was repaired. The save-prompt could also be unrelated to corruptions, but be of another nature, such as a Java script.

5.3.3 What is the Difference between Repair and Recover?

The 3-Heights™ PDF Analysis & Repair API can repair virtually everything it can detect. It can however not recover lost information. E.g. if a PDF document was sent via E-Mail attachment and only half of the attachment was sent before the connection was cut, information is lost. If information is lost, a document cannot be transformed back to its original state. In this case the document can only be recovered, meaning all the remaining information contained in the PDF document is recovered and used to create a new, valid PDF, however the new PDF is different from the original.

If a document contains syntactic or semantic faults which can be detected and fixed, the document can be repaired. A trivial example for such a case is image which contains image data with a length of 100 bits, the Length attribute of the image object however states a different, incorrect value, then this value can be corrected and the document can be repaired.
5.4 Concepts

5.4.1 Analysis Only

The process for analysis-only is the most simple process that can be implemented by the API. The steps in this process are shown in the graphic below:

- A new PDFRepair object is created.
- A PDF input-document is opened using the function `Open`.
- Analysis-options are set using the property `AnalysisOptions`. This step is optional.
- The analysis of the input-document is performed using the function `Analyze`.
- A list of error objects can be retrieved using the functions `GetFirstError` and consecutive calls to the function `GetNextError` until no more errors are returned. An error objects provides information about a corruption error, such as an error code or an error message.
- The input-document is then closed again using the function `Close`.

Call sequence for analysis-only

- Create object
- Open
- AnalysisOptions
- Analysis
- Report errors
- Close

Example: A simplified Visual Basic 6 program with the above call sequence looks about as shown below

```vbnet
dim repair as new PDFREPAIRAPILib.PDFRepair
repair.Open(...) 
repair.AnalysisOptions = ...
repair.Analysis()
' Report Errors

dim err as PDFREPAIRAPILib.PdfError
set err = repair.GetFirstError
while (not err is nothing)
  ' Do something with the error, e.g. output err.message
...
  set err = repair.GetNextError
wend
repair.Close()
```

A more detailed and executable Visual Basic 6 sample is provided with the release as well as with the evaluation version.
5.4.2 Analysis & Repair

Often corrupt documents not only need to be detected, but also repaired or recovered. As opposed to the analysis-only process, here the file is analyzed and repaired in one step using the function `AnalyzeAndRepair`. The repaired document is saved as a new document using the function `SaveAs`.

Call sequence for analysis and repair
- Create Object
- Open
- Set `AnalysisOptions` and `RecoveryOptions`
- `AnalyzeAndRepair`
- `SaveAs`
- `Report Errors`
- `Close`

5.4.3 Analysis & Conditional Repair

In the Analysis & Repair process using `AnalyzeAndRepair`, every document is repaired, regardless of the analysis. A more sophisticated approach is to separate these two steps to first analyze the document and only repair it if corruptions are actually detected.

Call sequence for analysis and conditional repair
- Create Object
- Open
- Set `AnalysisOptions` and `RecoveryOptions`
- `Analysis`
- If errors
  - `Report Errors`
  - `Repair`
  - `SaveAs`
- `Close`

5.4.4 How to use the in-Memory Functions

The 3-Heights™ PDF Analysis & Repair API always requires a PDF input-document from which it reads and optionally a PDF output-document to where the result is saved to.
To open and save to from files, the functions `Open` and `SaveAs` are used.

Instead of accessing files, the PDF documents can be read and written to in-memory. The corresponding functions are `OpenMem` and `SaveInMemory`.

Once the output-document is saved to memory using `SaveInMemory`, that memory block can be accessed using the function `GetPdf`.

A call sequence to create a first PDFRepair Object that opens a PDF from file and stores its output in-memory and then a second object, which reads that in-memory document and saves it back to a file looks like this:

```csharp
PDFRepair1.Open(InputFile)
PDFRepair1.SaveInMemory()
PDFRepair1.Close()
PDFRepair2.OpenMem(PDFRepair1.GetPdf())
PDFRepair2.SaveAs(OutputFile)
PDFRepair2.Close()
```

This call sequence of course does not make much sense, it’s merely used to illustrate how to use of the in-memory functions. In a real application, the in-memory document is read from another application or a database.

### 5.5 Error Handling

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

Methods communicate their level of success or failure using their return value. Which return values have to be interpreted as failures is documented in the chapter Interface Reference. To identify the error on a programmatic level, check the property `ErrorCode`. The property `ErrorMessage` provides a human readable error message, describing the error.

**Example:**

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

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

6.1 PDFRepair Interface

6.1.1 AnalysisOptions

<table>
<thead>
<tr>
<th>Property (get, set):</th>
<th>TPDFAnalysisOption <code>AnalysisOptions</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td><code>eAnalyzeObjects + eAnalyzePageTree + eAnalyzeContentStreams</code></td>
</tr>
</tbody>
</table>

This property sets the analysis options. Options can be turned off to increase the speed of the analysis.

See enumeration `TPDFAnalysisOption`.

6.1.2 Analyze

**Method:** Boolean `Analyze()`

This method analyzes the input document for errors.

**Returns:**

- **True**  The document was successfully analyzed and no errors or warnings were issued.
- **False** Either the document could not be analyzed (`PDF_E_STOPPED`) or the analysis has reported errors or warnings. Use the `GetFirstError` and `GetNextError` methods for more information on the kind of problems encountered.

6.1.3 AnalyzeAndRepair

**Method:** Boolean `AnalyzeAndRepair()`

This method analyzes the input document and creates a repaired output document.

**Returns:**

- **True**  The document was successfully analyzed and a repaired document could be created.
The document could not be analyzed or repaired (PDF_E_FATAL).

6.1.4 Close

<table>
<thead>
<tr>
<th>Method:</th>
<th>Boolean Close()</th>
</tr>
</thead>
</table>

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

Returns:

True The file was closed successfully.
False Otherwise.

6.1.5 Diagnosis

<table>
<thead>
<tr>
<th>Property (get):</th>
<th>TPDFDiagnosis Diagnosis</th>
</tr>
</thead>
</table>

This property returns the diagnosis flags. See also enumeration TPDFDiagnosis.

6.1.6 ErrorCode

<table>
<thead>
<tr>
<th>Property (get):</th>
<th>TPDFErrorCode ErrorCode</th>
</tr>
</thead>
</table>

This property can be accessed to receive the latest error code. This value should only be read if a function call on the PDF Analysis & Repair API has returned a value, which signal a failure of the function (see chapter Error Handling). See also enumeration TPDFErrorCode. PDF-Tools error codes are listed in the header file bseerror.h. Please note that only few of them are relevant for the 3-Heights™ PDF Analysis & Repair API.

6.1.7 ErrorLevel

<table>
<thead>
<tr>
<th>Property (get):</th>
<th>Integer ErrorLevel</th>
</tr>
</thead>
</table>

This property can be accessed to check whether no errors (0), warnings only (1) or errors (2) were found during the analysis.

This property should be got after Analyze.

6.1.8 ErrorMessage

<table>
<thead>
<tr>
<th>Property (get):</th>
<th>String ErrorMessage</th>
</tr>
</thead>
</table>
Return the error message text associated with the last error (see property Error Code). This message can be used to inform the user about the error that has occurred. This value should only be read if a function call on the PDF Analysis & Repair API has returned a value, which signals a failure of the function (see chapter Error Handling).

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

### 6.1.9 GetFirstError

**Method:** PdfError GetFirstError()

This method returns the first error, it can also be a warning.

**Returns:**

The first error if there are any. Nothing otherwise.

### 6.1.10 GetNextError

**Method:** PdfError GetNextError()

This method returns the next error, it can also be a warning.

**Returns:**

The next error if there is any. Nothing otherwise.

### 6.1.11 GetPdf

**Method:** Variant GetPdf()

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

**Returns:**

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

**Property (get):** Boolean LicenseIsValid

Check if the license is valid.

6.1.13 Open

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

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

**Parameters:**

- **Filename** [String] The file name and optionally the file path, drive or server string according to the operating systems file name specification rules.
- **Password** [String] (optional) The user or the owner password of the encrypted PDF document. If this parameter is left out an empty string is used as a default.

**Returns:**

- **True** The file could be successfully opened.
- **False** The file does not exist, it is corrupt, or the password is not valid. Use the properties ErrorCode and ErrorMessage for additional information.

6.1.14 OpenMem

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

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

**Parameters:**

- **MemBlock** [Variant] The memory block containing the PDF file given as a one dimensional byte array.
- **Password** [String] (optional) The user or the owner password of the encrypted PDF document. If this parameter is left out an empty string is used as a default.

**Returns:**

- **True** The document could be successfully opened.
6.1.15 ProductVersion

**Property (get):** String ProductVersion

Get the version of the 3-Heights™ PDF Analysis & Repair API in the format “A.C.D.E”.

6.1.16 RebuildOptions

**Property (get, set):** TPDFRebuildOption RebuildOptions

Default: 0

This property can be used to get or set the rebuild options.

See TPDFRebuildOption.

6.1.17 RecoveryOptions

**Property (get, set):** TPDFRecoveryOption RecoveryOptions

Default: eRecoverXREF + eRecoverPages

This property can be used to get or set the recovery options.

See TPDFRecoveryOption.

6.1.18 Repair

**Method:** Boolean Repair(String InputFile, String OutputFile, String LogFile)

This method opens a PDF file, analyzes and repairs it. The repaired file is saved with a new file name. Optionally a log file can be generated.

**Parameters:**

- **String [InputFile]** The file name and optionally the file path, drive or server string according to the operating systems file name specification rules of the input file.

- **String [OutputFile]** The file name and optionally the file path, drive or server string of the output file (the repaired file).

- **String [LogFile] (optional)** The path to the log-file.
Returns:

True  The file could successfully be processed.
False  The file could not be processed and therefore not be repaired.

6.1.19 ReportingLevel

<table>
<thead>
<tr>
<th>Property (get, set):</th>
<th>Integer ReportingLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>3</td>
</tr>
</tbody>
</table>

With this property the reporting level can be set or got. The supported levels are:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>none Nothing is reported</td>
</tr>
<tr>
<td>1</td>
<td>errors Errors are reported</td>
</tr>
<tr>
<td>2</td>
<td>warnings Errors and warnings are reported</td>
</tr>
<tr>
<td>3</td>
<td>information Error, warnings and information are reported</td>
</tr>
</tbody>
</table>

The property ReportingLevel must be set before the Open method in order to be applied.

6.1.20 SaveAs

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

Save the currently opened document.

Parameters:

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

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

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

PermissionFlags  [TPDFPermission]  (optional) The permission flags.

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

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

**False** Otherwise. One of the following occurred:
- The output file cannot be created.
- *PDF_E_FILECREATE*: Failed to create the file.

### 6.1.21 SaveInMemory

**Method:** Boolean *SaveInMemory()*

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

**Returns:**

**True** The document could be saved in memory successfully.

**False** Otherwise.

### 6.1.22 SetLicenseKey

**Method:** Boolean *SetLicenseKey(String LicenseKey)*

Set the license key.

### 6.2 PdfError Interface

#### 6.2.1 Count

**Property (get):** Long *Count*

This property returns how many times the error occurs on the page.

#### 6.2.2 ErrorCode

**Property (get):** TPDFErrorCode *ErrorCode*

---

This is not a complete list. If *SaveAs* returns *False*, it is recommended to abort the processing of the file and log the error code and error message.
This property can be accessed to receive the latest error code. This value should only be read if a function call on the PDF Analysis & Repair API has returned a value, which signals a failure of the function (see chapter Error Handling). See also enumeration TPDFErrorCode. PDF-Tools error codes are listed in the header file bseerror.h. Please note that only few of them are relevant for the 3-Heights™ PDF Analysis & Repair API.

### 6.2.3 Message

**Property (get):** String Message

This property returns an explaining error message.

### 6.2.4 ObjectNo

**Property (get):** Long ObjectNo

This property returns the object number at which the error occurs. If the error is not related to a particular object, 0 is returned.

### 6.2.5 PageNo

**Property (get):** Long PageNo

This property returns the page number on which the error occurs. If the error is not related to a particular page number, 0 is returned.

### 6.3 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).

#### 6.3.1 TPDFAnalysisOption Enumeration

- eAnalyzeObjects: Analyze objects
- eAnalyzePageTree: Analyze page tree
- eAnalyzeContentStreams: Analyze content streams

#### 6.3.2 TPDFDiagnosis Enumeration

- eDiagnosisOpen: Diagnose opening
6.3.3 TPDFErrorCode Enumeration

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

The single letter gives an indication of the severity of the error. These are: Success, Error, Warning and Information.

In general, an error is returned if an operation could not be completed, e.g. no valid output file was created. A warning is returned if the operation was completed, but problems occurred in the process.

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

<table>
<thead>
<tr>
<th>TPDFErrorCode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF_S_SUCCESS</td>
<td>The operation was completed successfully.</td>
</tr>
<tr>
<td>LIC_E_NOTINIT, ...</td>
<td>Various license management related errors.</td>
</tr>
<tr>
<td>LIC_E_LEVEL</td>
<td></td>
</tr>
<tr>
<td>PDF_E_FILEOPEN</td>
<td>Failed to open the file.</td>
</tr>
<tr>
<td>PDF_E_FILECREATE</td>
<td>Failed to create the file.</td>
</tr>
<tr>
<td>PDF_W_NOENCRYPTION</td>
<td>The file is PDF/A and must not be encrypted.</td>
</tr>
</tbody>
</table>

6.3.4 TPDFPermission Enumeration

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

<table>
<thead>
<tr>
<th>TPDFPermissionFlag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ePermPrint</td>
<td>Low resolution printing</td>
</tr>
<tr>
<td>ePermModify</td>
<td>Changing the document</td>
</tr>
<tr>
<td>ePermCopy</td>
<td>Content copying or extraction</td>
</tr>
<tr>
<td>ePermAnnotate</td>
<td>Annotations</td>
</tr>
<tr>
<td>ePermFillForms</td>
<td>Filling of form fields</td>
</tr>
<tr>
<td>ePermSupportDisabilities</td>
<td>Support for disabilities</td>
</tr>
<tr>
<td>ePermAssemble</td>
<td>Document assembly</td>
</tr>
</tbody>
</table>
**TPDFPermission Table**

| ePermDigitalPrint | High resolution printing |

Changing permissions or combining multiple permissions is done using a bitwise “or” operator.

Changing the current permissions in Visual Basic should be done like this:

**Allow Printing**

\[
\text{Permission} = \text{Permission} \text{ Or } \text{ePermPrint}
\]

**Prohibit Printing**

\[
\text{Permission} = \text{Permission} \text{ And Not } \text{ePermPrint}
\]

### 6.3.5 TPDFRebuildOption Enumeration

- **eRebuildStreams** Recompress all streams.
  - This setting is recommended for files with corrupt streams.

- **eRebuildFonts** Rebuild fonts.

- **eRebuildFontsAsType1** Convert Compact Font Format (CFF) simple fonts to Type1.
  - This property may be used together with eRebuildFonts only.

### 6.3.6 TPDFRecoveryOption Enumeration

- **eRecoverXREF** Recover the XREF table. Disabling this option might be useful if processing a document takes too long, since repairing the cross-reference table is very time consuming.

- **eRecoverPages** If pages are not part of the page tree (loose pages), they will be recovered and added at the end of the document. If they should not be recovered, these pages will be removed from the document.

- **eRecoverImages** Deprecated in Version 4.7, this option has no effect and will be removed in future versions.
7 Version History

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

7.1 Changes in Version 6

- [Java] Changed minimal supported Java language version to 7 (previously 6).
- [PHP] Removed all versions of the PHP interface.
- [.NET] New availability of this product as NuGet package for Windows, macOS and Linux.
- [.NET] New support for .NET Core versions 1.0 and higher. The support is restricted to a subset of the operating systems supported by .NET Core, see Operating Systems.
- [.NET] Changed platform support for NuGet packages: The platform "AnyCPU" is now supported for .NET Framework projects.

7.2 Changes in Version 5

- Changed error reporting behavior: Errors in the XMP metadata are no longer reported when saving a recovered document.
- [PHP] New extension PHP 7.3 (non thread safe) for Linux.

7.3 Changes in Version 4.12

- New HTTP proxy setting in the GUI license manager.

7.4 Changes in Version 4.11

- New support for reading and writing PDF 2.0 documents.
- New support for the creation of output files larger than 10GB (not PDF/A-1).
- New treatment of the DocumentID. In contrast to the InstanceID the DocumentID of the output document is inherited from the input document.
- [PHP] New Interface for Windows and Linux. Supported versions are PHP 5.6 & 7.0 (Non Thread Safe). The PdfRepairAPI PHP Interface is contained in the 3-Heights™ PDF Tools PHP5.6 Extension and the 3-Heights™ PDF Tools PHP7.0 Extension.
- [C] Changed 32-bit binaries on Windows that link to the API need to be recompiled due to a change of the used mangling scheme.

7.5 Changes in Version 4.10

- Improved robustness against corrupt embedded font files.
- Improved robustness against corrupt input PDF documents.
- [C] Clarified Error handling of TPdfStreamDescriptor functions.
7.6 Changes in Version 4.9

- **Improved** support for recovering certain corruption types.
- **Improved** support for and robustness against corrupt input PDF documents.
- **Improved** repair of embedded font programs that are corrupt.
- **New** support for OpenType font collections in installed font collection.
- **Improved** metadata generation for standard PDF properties.
- [C] **Changed** return value `pfGetLength` of `TPDFStreamDescriptor` to `pos_t`.8

7.7 Changes in Version 4.8

- **Added** repair functionality for TrueType font programs whose glyphs are not ordered correctly.
- [.NET] **Deprecated** method `GetLicenseIsValid`.
- [.NET] **New** property `LicenseIsValid`.8

---

8 This has no effect on neither the .NET, Java, nor COM API
8 Licensing, Copyright, and Contact

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

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

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

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