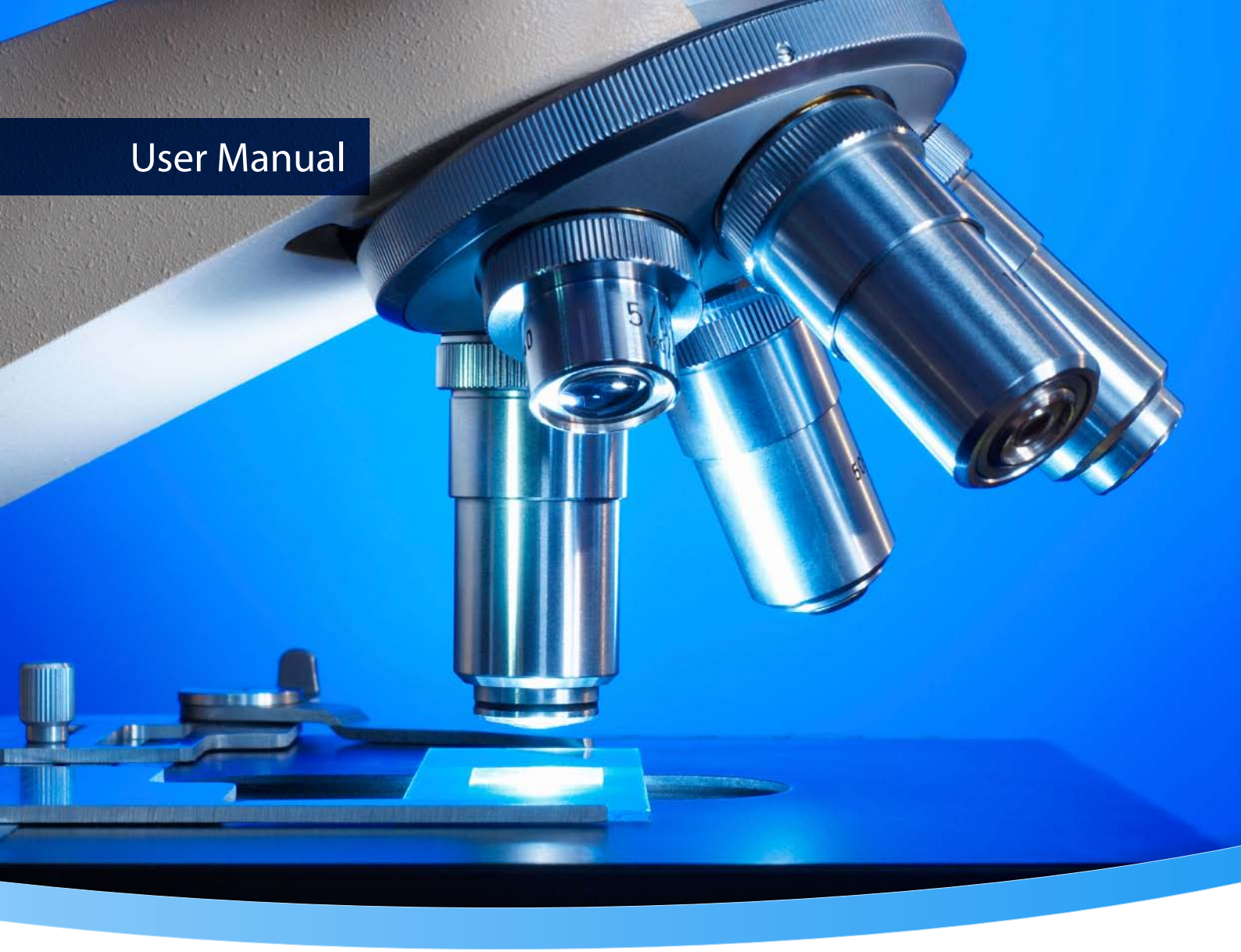
User Manual



3-Heights®
PDF Validator API

PDF/A conform

Version 6.27.6

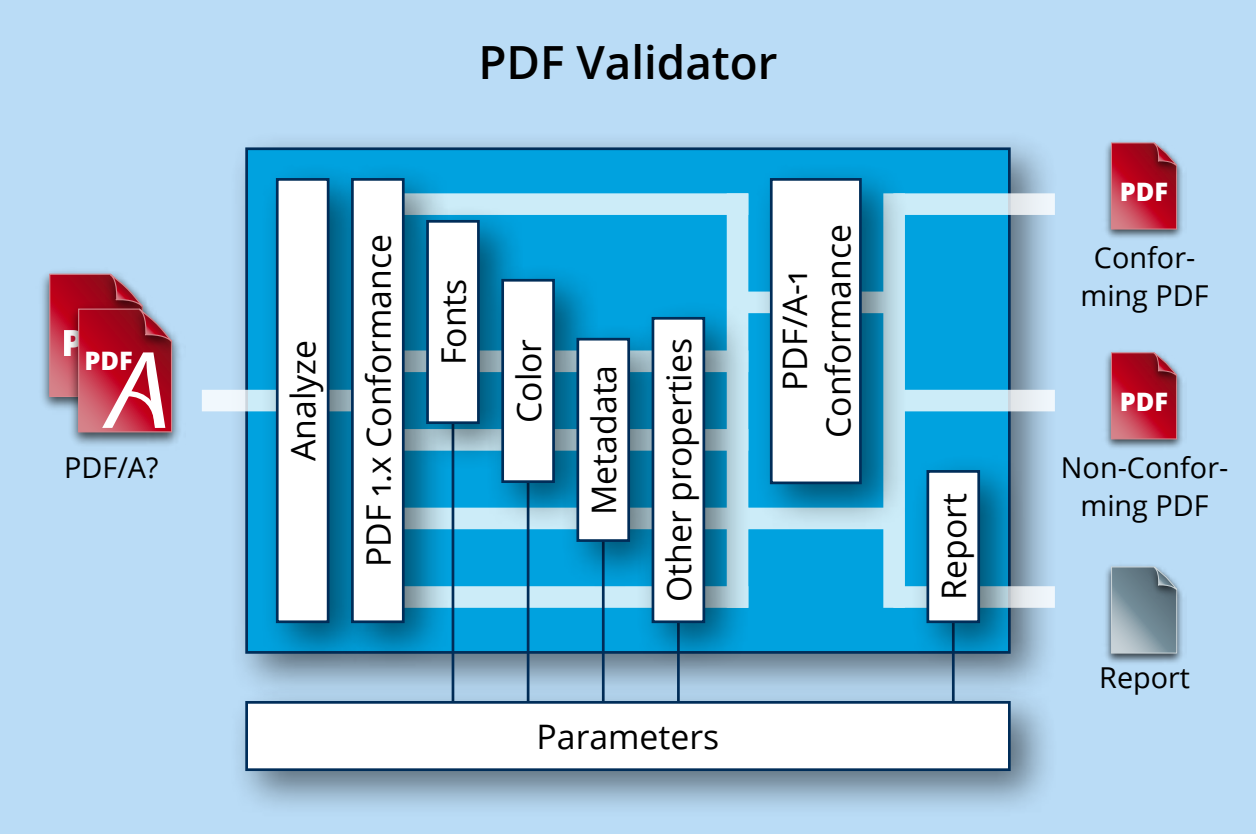
pdftools

Contents

1 Introduction

1.1 Description

The 3-Heights® PDF Validator API safeguards the quality of PDF documents. It checks PDF files for conformance to the ISO standards for PDF and PDF/A documents. Unfortunately, there are many PDF creation or manipulation tools in use that do not comply with the PDF or PDF/A standard. System and operational interruptions often occur as a result. Incoming documents should be verified before they flow into business processes to prevent interruptions of this nature and to avoid unexpected costs.



The 3-Heights® PDF Validator API checks whether PDF documents comply with the PDF or PDF/A standard. Additional verification tests such as checking the version number of the PDF document are also possible; the tool can also verify conformance to internal directives - use of the right color, for instance, or use of the right fonts and other specifications.

Through its interfaces (C, Java, .NET, COM) and thanks to its flexibility, developers can integrate the 3-Heights® PDF Validator API in virtually any application.

1.2 Functions

3-Heights® PDF Validator API verifies PDF documents in accordance with the ISO standard for PDF and also PDF/A for long-term archiving. The tool can check the conformity of individual documents and entire archives. The result output is needs-oriented, e.g. a detailed report for a manufacturer of PDF software or a summary of error reports for the user. The description includes every detail such as frequency, page number, or PDF object number. Verification of internal specifications (e.g. standard image resolution) can occur at the same time.

1.2.1 Features

- PDF document validation on the basis of various PDF specifications (PDF 1.x, PDF 2.0, PDF/A-1, PDF/A-2, PDF/A-3)
- PDF-conforming dependent lexical, syntactic, and semantic checks (see Coverage)
- Detailed or summarized reporting (log file)
- Detailed error description (number, type, description, PDF object, page number)
- Classification by error, warning and information
- Optional cancellation of validation on occurrence of the first error
- Reading of encrypted PDF files
- Determination of claimed conformance of document
- Validatation of conformance to corporate directives defined in custom profile
- Reading of input document from file, memory, or stream

1.2.2 Formats

Input Formats:

- PDF 1.x (PDF 1.3, …, PDF 1.7)
- PDF 2.0
- PDF/A-1a, PDF/A-1b
- PDF/A-2a, PDF/A-2b, PDF/A-2u
- PDF/A-3a, PDF/A-3b, PDF/A-3u

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)
- Quality assurance: veraPDF test corpus and Isartor test suite

1.3 Interfaces

The following interfaces are available:

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

[1] Limited supported OS versions. Operating systems

1.4 Operating systems

The 3-Heights® PDF Validator 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
- 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 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.

2 Installation and deployment

2.1 Windows

The 3-Heights® PDF Validator 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 Validator 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.

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 Validator API provides two interfaces:

- Java interface
- Native C interface

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

File description

Name	Description
bin/x64/libPdfValidatorAPI.so	Shared library that contains the main functionality. The file's extension differs on macOS, (.dylib instead of .so).
doc/*.*	Documentation
include/*.h	Header files to include in your C/C++ project
jar/VALA.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 libPdfValidatorAPI.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/libPdfValidatorAPI.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.

2.2.2 macOS

The shared library must have the extension .jnilib for use with Java. Create a file link for this purpose by using the following command:

```
ln libPdfValidatorAPI.dylib libPdfValidatorAPI.jnilib
```

2.3 ZIP archive

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

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

2.3.1 Development

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

Req. The file is required for this interface.

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

Doc. The file is for documentation only.

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

Files for development

Name	.NET	Java	COM	C
bin\‹platform›\PdfValidatorAPI.dll	Req.	Req.	Req.	Req.
bin*NET.dll	Req.			
bin*NET.xml	Doc.			
doc*.pdf	Doc.	Doc.	Doc.	Doc.
doc\PdfValidatorAPI.idl			Doc.	
doc\javadoc*.*		Doc.		

Files for development

Name	.NET	Java	COM	C
include\pdfvalidatorapi_c.h				Req.
include*.*				Opt.
jar\VALA.jar		Req.		
lib\‹platform›\PdfValidatorAPI.lib				Req.[2]
samples*.*	Doc.	Doc.	Doc.	Doc.

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

File description

Name	Description
bin\‹platform›\PdfValidatorAPI.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.
doc*.*	Documentation
include*.*	Files to include in your C / C++ project
lib\‹platform›\PdfValidatorAPI.lib	On Windows operating systems, the object file library needs to be linked to the C/C++ project.
jar\VALA.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 four different interfaces.

Files for deployment

Name	.NET	Java	COM	C
bin\‹platform›\PdfValidatorAPI.dll	Req.	Req.	Req.	Req.

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

Files for deployment

`bin\*NET.dll`	Req.
`jar\VALA.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).

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 `PdfValidatorAPI.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 PdfValidatorAPI.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%\PdfValidatorAPI.dll.
```

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 Validator API contains all the libraries needed, both managed and native.

Installation

The package `PdfTools.PdfValidator 6.27.6` 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.PdfValidator 6.27.6`.

Development

The package `PdfTools.PdfValidator 6.27.6` contains .NET libraries with versions .NET Standard 1.1, .NET Standard 2.0, and .NET Framework 2.0, and native libraries for Windows, macOS, 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 Validator 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 COM interface

Registration

Before you can use the 3-Heights® PDF Validator API component in your COM application program, you have to register the component using the `regsvr32.exe` program that is provided with the Windows operating system. The following command shows how to register the PdfValidatorAPI.dll. In Windows Vista and later, the command needs to be executed from an administrator shell.

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

2.5.2 Java interface

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

For compilation and execution

When using the Java interface, the Java wrapper `jar\VALA.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\VALA.jar" ^
 sampleApplication.java
```

For execution

Additionally, the library `PdfValidatorAPI.dll` needs be in one of the system's library directories[5] 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[6].

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

[5] On Windows defined by the environment variable PATH, and 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`"

```
java -classpath ".;C:\Program Files\PDF Tools AG\VALA.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/VALA.jar" ...
```

2.5.3 .NET interface

The 3-Heights® PDF Validator 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. `PdfValidatorAPI.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 `PdfValidatorAPI.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 `PdfValidatorAPI.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 `PdfValidatorAPI.dll` are copied to subdirectories `win-x86` and `win-x46` of the output directory, respectively.

2.5.4 C interface

- The header file `pdfvalidatorapi_c.h` needs to be included in the C/C++ program.
- On Windows operating systems, the library `PdfValidatorAPI.lib` needs to be linked to the project.
- The dynamic link library `PdfValidatorAPI.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.

3 License management

The 3-Heights® PDF Validator 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.

3.1 License features

The functionality of the 3-Heights® PDF Validator API contains one area to which the following license feature is assigned:

`Custom` Verify conformance to custom corporate directives.

The presence of this feature in a given license key can be checked in the license manager. The Interface reference specifies in more detail which functions are included in this license feature.

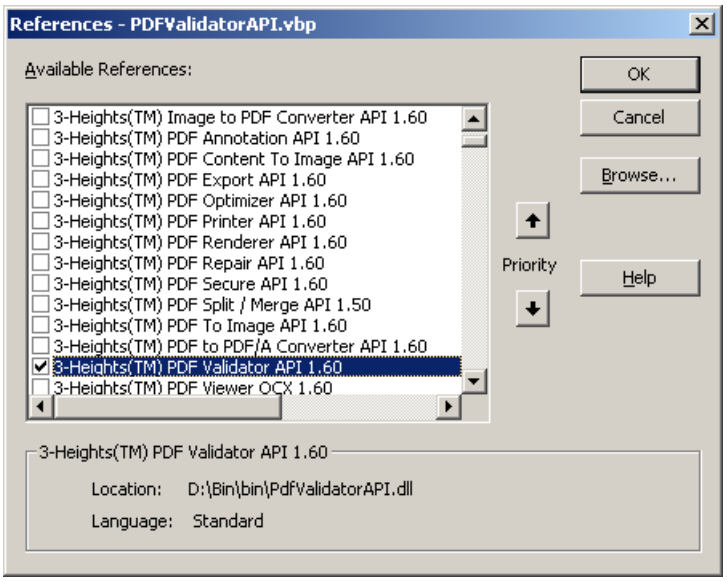
4 Programming interfaces

4.1 Visual Basic 6

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

If you start from scratch, perform these steps:

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



2. Draw a new Command Button and optionally, rename it if you like.
3. Double-click the command button and insert the few lines of code below. All that you need to change is the path of the file name.

Example:

```
Dim validator As New PDFVALIDATORAPILib.PDFValidator
Dim err As PDFVALIDATORAPILib.PDFError
...
validator.Open(file, "", ePDFA1b)
validator.ReportingLevel = 2
validator.Validate
...
```

4.2 .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 Validator API. The easiest to quickly start is to refer to this sample.

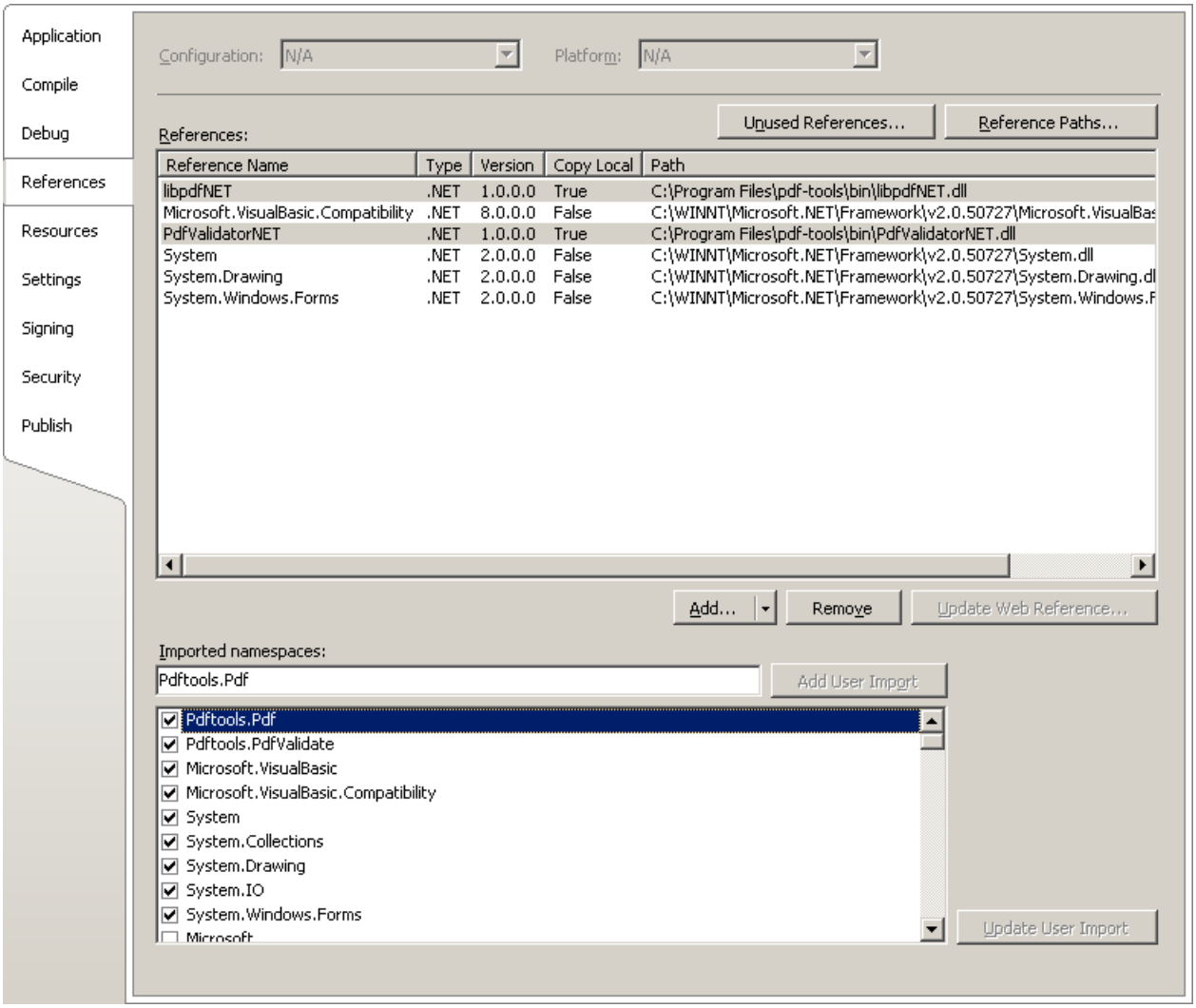
To create a new project from scratch, perform the following steps:

1. Start Visual Studio and create a new C# or VB project.
2. Add references to the NuGet package `PdfTools.PdfValidator 6.27.6`, as described in NuGet package.
3. Import namespaces (Note: This step is optional, but useful.)
4. Write your code.

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

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



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

Example:

```
Dim validator As New PdfValidator
Dim PDFVersion As PDFCompliance = PDFCompliance.ePDFA1b
```

```
Dim FileName, Password As String
...
validator.Open(FileName, Password, PDFVersion)
...
```

4.2.2 C#

3. Add the following namespaces:

Example:

```
using Pdftools.Pdf;
using Pdftools.PdfValidate;
```

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

Example:

```
using (PdfValidator validator = new PdfValidator())
{
    String FileName, Password;
    ...
    validator.Open(FileName, Password)
    ...
}
```

4.2.3 Deployment

This is a guideline on how to distribute a .NET project that uses the 3-Heights® PDF Validator 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 Validator API's .NET assemblies (`*NET.dll`), as well as the native library (`PdfValidatorAPI.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 `PdfValidatorAPI.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.

4.2.4 Troubleshooting: TypeInitializationException

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

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

`System.DllNotFoundException` Unable to load DLL `PdfValidatorAPI.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 `PdfValidatorAPI.dll` could not be found at execution time.

Resolve this by performing one of these actions:

- Use the NuGet package.
- Add `PdfValidatorAPI.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 `PdfValidatorAPI.dll` resides to the environment variable `%Path%`, or
- Manually copy `PdfValidatorAPI.dll` to the output directory of your project.

Troubleshooting: BadImageFormatException

The exception means that the native DLL `PdfValidatorAPI.dll` has the incorrect "bitness" (i.e. platform 32 vs. 64 bit). There are two versions of `PdfValidatorAPI.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.

4.3 Java

A Java sample `validate.java` is available which shows how to use the Java interface.

Example:

```
import com.pdftools.pdfvalidator.PdfValidatorAPI;
```

```
import com.pdftools.pdfvalidator.PdfError;
import com.pdftools.NativeLibrary;
...
PdfValidatorAPI doc = new PdfValidatorAPI();
doc.setReportingLevel(2);
doc.open(file, "", NativeLibrary.COMPLIANCE.ePDFA1b);
...
```

4.4 C

There is a C sample available within the software package of the evaluation and release version that shows how the C interface is used. Before the C interface can be used to create objects, it must be initialized once. This is done using `PdfValidatorInitialize`. To un-initialize, use `PdfValidatorUnInitialize`. Other than that, equal call sequences as in other interface can be used.

Example:

```
#include "pdfvalidatorapi_c.h"
TPdfValidator* pDocument;
TPdfValidatorError* pError;

PdfValidatorInitialize();
pDocument = PdfValidatorCreateObject();
PdfValidatorOpenA(pDocument, argv[1], "", ePDFA1b))
PdfValidatorSetStopOnError(pDocument, 0);
PdfValidatorValidate(pDocument);
...
PdfVlidatorUnInitialize();
```

5 User guide

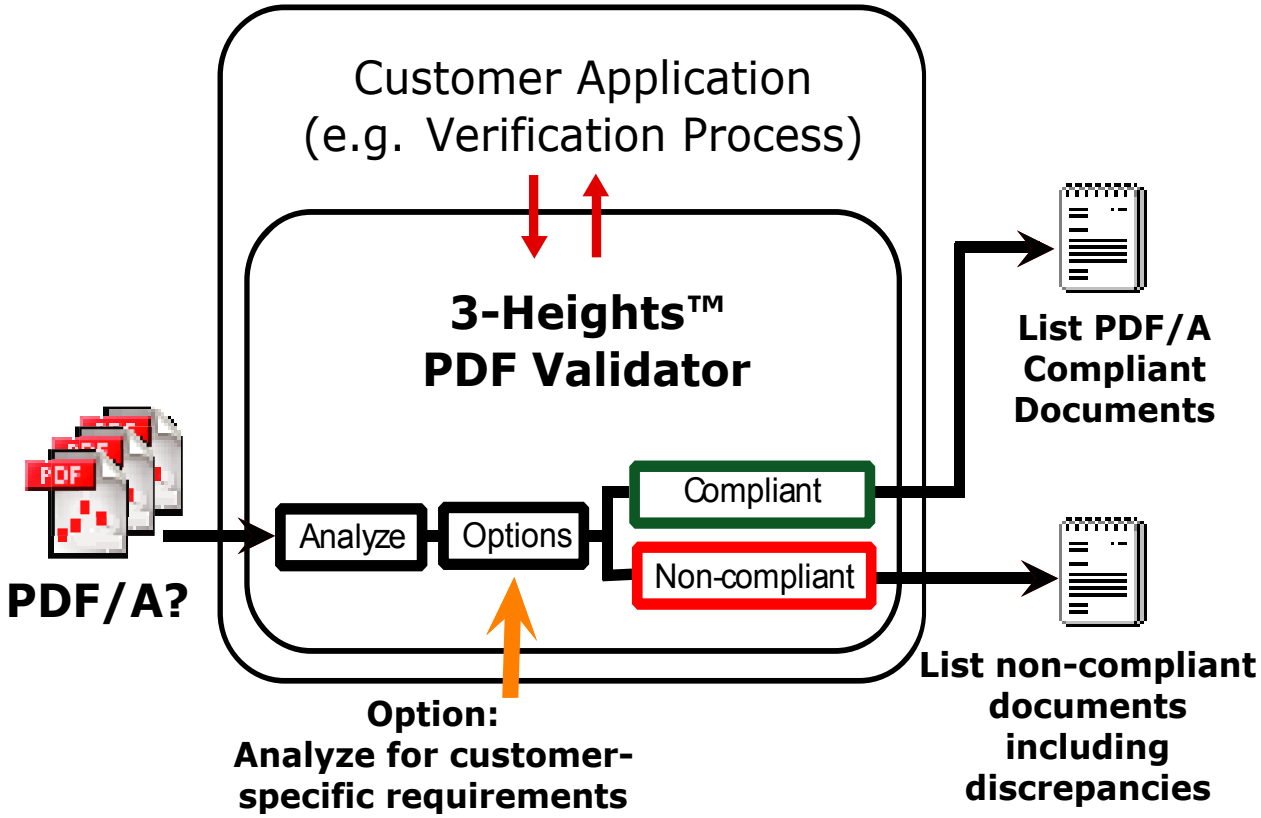
5.1 Overview of the API

5.1.1 About the 3-Heights® PDF Validator API

The 3-Heights® PDF Validator API is a tool to validate existing PDF documents against a specification, such as the international standard ISO 19005-1 for PDF/A. The tool analyzes a PDF document and states whether it conforms to a specification or not. If a document fails to conform, the tool provides detailed information about why the validation failed. This consists of either a list of all validation errors, including a brief error description, the page number, the PDF object number, and number of occurrences, or a summary.

5.2 About the API

The API requires as input a PDF document and the selection of a conformance level (e.g. PDF/A-1b).



1. The API opens a PDF document; at that point, a conformance level must be selected.
2. The reporting level decides what errors types are reported later (none, errors only, errors + warnings, errors + warnings + information).
3. The document is validated against the selected specification.
4. A list of all errors can be retrieved after the validation.
5. The document is closed.

Below is a call sequence writing in Visual Basic .NET, which illustrates this procedure:

Example:

```
Dim validator As New PdfValidator
validator.ReportingLevel = ...
validator.Open(...)
validator.Validate()
Dim PdfErr As PdfError = validator.GetFirstError
While Not (PdfErr Is Nothing)
    ' Do something with PdfErr, e.g.\ output PdfErr.Message
   PdfErr = validator.GetNextError
End While
validator.Close()
```

Note that the call sequence above is a bit too simple. If you are using this API for the first time, it may be best to look at one of the provided samples to start with.

The 3-Heights® PDF Validator API provides two ways to list conformance errors:

1. List all errors individually. This is the method used in the call sequence above. Every error can be listed including page number, PDF object, error code, and error message. Multiple equal errors on the same page are merged and the number of occurrences is provided. This approach lists very detailed information, which is useful for a creator of the PDF document.
2. Instead of listing all errors individually, it is possible to summarize them in 19 generic categories. For example, in a PDF documentm if all conformance errors are related to non-embedded fonts, only one category is listed. An end user is most likely not interested in a detailed list, but instead only wants to know whether the validated document conforms or not. If additional information is not required, a summary is sufficient.

5.2.1 Using the customized extensions

If you purchased a customized version of the 3-Heights® PDF Validator API with customized features (such as validate if embedded images have a resolution within a specified range), see additional documentation `vala_custom_extensions_*.pdf`.

5.3 What is PDF/A?

PDF/A is an ISO standard for using the PDF format for the long-term archiving of electronic documents. This chapter provides a brief overview. For additional information, see https://www.pdf-tools.com/en/resources/pdf-iso-standards/.

5.3.1 PDF/A-1

PDF/A-1 (ISO 19005-1) is based on PDF 1.4 (Acrobat 5). On top of PDF 1.4, it has additional requirements to keep the document self-contained and suitable for long-term archival. The most important are:

- Encryption may not be used
- If device-dependent color space (e.g. DeviceRGB, DeviceCMYK, DeviceGray) are used, a corresponding color profile must be embedded
- Fonts used for visible text must be embedded
- Transparency may not be used

5.3.2 PDF/A-2

PDF/A-2 is described in ISO 19005-2. It is based on ISO 32000-1, the standard for PDF 1.7. PDF/A-2 is meant as an extension to PDF/A-1. The second part complements the first part and does not replace it. The most important differences between PDF/A-1 and PDF/A-2 are:

- The list of compression types has been extended by JPEG2000
- Transparent contents produced by graphic programs are allowed
- Optional contents (also known as layers) can be made visible or invisible
- Multiple PDF/A files can be bundled in one file (collection, package)
- The additional conformity level U (Unicode) allows for creating searchable files without having to fulfill the strict requirements of the conformity level A (accessibility)
- File size can be reduced using compressed object and XRef streams

Documents that contain features described above, in particular layers or transparency, should therefore be converted to PDF/A-2 rather than PDF/A-1.

5.3.3 PDF/A-3

PDF/A-3 is described in ISO 19005-3. It is based on ISO 32000-1, the standard for PDF 1.7. PDF/A-3 is an extension to PDF/A-2. The third part shall complement the second part and not replace it. The only two differences between PDF/A-2 and PDF/A-3 are:

- Files of any format and conformance may be embedded. Embedded files need not be suitable for long-term archiving.
- Embed files can be associated with any part of the PDF/A-3 file.

5.4 Error, warning, and information

Error codes in the 3-Heights® PDF Validator API are classified in three types. The meaning of these three types with respect to validation is described below:

5.4.1 Information

A message of type “information” describes a process step performed by the program. Examples:

- A hint about the next step that is going to be performed
- A detection that a document does not follow a recommendation of a specification

A message of “information” type does not indicate a problem or violation of a specification. No action is required.

5.4.2 Warning

A warning indicates a violation of the PDF/A specification. A typical warning is formatting error, such as a missing, but required entry or a prohibited entry. The document may still conform to PDF, but not to PDF/A.

A PDF document that raises warnings but no errors is likely to be recoverable e.g. using the 3-Heights® PDF to PDF/A Converter. No critical data is missing. The document, even though not conforming to PDF/A, is still worthwhile.

5.4.3 Error

An error indicates a violation of the PDF specification or a severe violation of the PDF/A specification. An error is more severe than a warning. A typical error is a corruption, such as missing or invalid data. A PDF document, which

raises an error, is likely to be not fully recoverable. Critical data might be missing. An error can sometimes be repaired using the 3-Heights® PDF to PDF/A Converter. An error however often indicates that data is missing. Depending on the type of data, a PDF to PDF/A converter may or may not be able to restore the data adequately. Examples:

- Repairable: If a color profile is missing or invalid, it can be replaced by a new, correct color profile.
- Not repairable: If image data is missing, the image data cannot be repaired, it must be retrieved from the original image.

5.5 Custom validation profiles

In addition to checking documents' conformance to the PDF Reference and PDF ISO standards, the 3-Heights® PDF Validator API can ensure conformance to custom corporate directives. Custom checks are defined in a configuration file and activated using the `SetProfile, SetProfileMem, SetProfileStream` method.

The format of the configuration file follows the INI file syntax. By default, all custom checks are deactivated, so all custom checks must be enabled explicitly. All lines starting with a semicolon "`;`" are ignored.

5.5.1 `[File]` INI-File Section

`FileSize1`

Key: `FileSize1`
Error code: `CHK_E_FILESIZE1`

Define the maximum allowed file size in megabytes.

Example: Set allowed file size to 100 MB.

```
[File]
FileSize1=100
```

`FileSize2`

Key: `FileSize2`
Error code: `CHK_E_FILESIZE2`

Define a second limit for the maximum allowed file size in megabytes. If `FileSize2` is specified, it must be larger than the value of `FileSize1`. If a file's size is larger than `FileSize2`, the error `CHK_E_FILESIZE2` is raised; otherwise, if the size is larger than `FileSize1`, `CHK_E_FILESIZE1` is raised.

Example: Set allowed file size to 200 MB.

```
[File]
FileSize2=200
```

MaxPdfVersion

Key: MaxPdfVersion
Error code: CHK_E_MAXPDFVERS

The highest PDF version that a document may have is defined by the MaxPdfVersion setting. The argument is a period-separated value with a major version, a minor version, and an optional extension level.

Example: Set maximum allowed PDF version to PDF 1.4 (Acrobat 5).

```
[File]
MaxPdfVersion=1.4
```

Example: Set the maximum allowed PDF version to PDF 1.7, extension level 3 (Acrobat 9).

```
[File]
MaxPdfVersion=1.7.3
```

MinPdfVersion

Key: MinPdfVersion
Error code: CHK_E_MINPDFVERS

The setting MinPdfVersion sets the minimum PDF version the document must have. The usage is equivalent to MaxPdfVersion.

Example: The following setting requires the document under test to be at least PDF 1.3 and no higher than PDF 1.6.

```
[File]
MinPdfVersion=1.3
MaxPdfVersion=1.6
```

Encryption

Key: Encryption
Error code: CHK_E_ENCRYPTION

Check whether or not the file is encrypted.

true Raise error if file is not encrypted.

false Raise error if file is encrypted.

Example: Dis-allow encrypted files.

```
[File]
Encryption=false
```

Linearization

Key: Linearization
Error code: CHK_E_LINEARIZATIION

Check whether or not the file is linearized.

true Raise error if file is not linearized.

false Raise error if file is linearized.

Example: Dis-allow linearized files.

```
[File]
Linearization=false
```

NonFilters, NonFilter‹i› (Non-approved filters)

Key: NonFilters

Key: NonFilter‹i›
Error code: CHK_E_FILTER

Non-approved stream filters are defined by setting NonFilters=‹n›, where ‹n› is the count of non-approved stream filters, i.e. a value larger than 0. The names of the filters are defined using NonFilter‹i›=‹Name i› where ‹i› is an index ranging from 1 to ‹n›. Possible values for ‹Name i› are the PDF filters:

- ASCIIHexDecode
- ASCII85Decode
- LZWDecode
- FlateDecode
- RunLengthDecode
- CCITTFaxDecode
- JBIG2Decode
- DCTDecode
- JPXDecode

Example: Disallow JPEG2000 compressed images:

```
[File]
NonFilters=1
NonFilter1=JPXDecode
```

5.5.2 [Document] INI-File Section

NonCreators, NonCreator‹i› (Non-approved PDF creators)

Key: NonCreators

Key: NonCreator‹i›
Error code: CHK_E_CREATOR

Non-approved PDF creators are defined by setting NonCreator=‹n›, where ‹n› is the count of non-approved creators, i.e. a value larger than 0. The names of the creators are defined using NonCreator‹i›=‹Name i›, where ‹i› is an index ranging from 1 to ‹n› and ‹Name i› is the name of the non-approved PDF creator.

Example: A list of non-approved PDF creators can be defined like this:

```
[Document]
NonCreators=2
NonCreator1=pdf fools
NonCreator2=badpdfcreator
```

NonProducers, NonProducer‹i› (Non-approved PDF producers)

Key: NonProducers

Key: NonProducerX
Error code: CHK_E_PRODUCER

Non-approved PDF producers are defined similar to non-approved PDF creators.

Example: A list of non-approved PDF producers can be defined like this:

```
[Document]
NonProducers=1
NonProducer1=pdf fools
```

EmbeddedFiles, EmbeddedFile‹i› (Allowed embedded file types)

Key: EmbeddedFiles

Key: EmbeddedFileX
Error code: CHK_E_EFTYPE

List of allowed embedded file types. Wildcards are supported at the beginning or the end of the string.

Example: Allow embedded PDF files and job options only.

```
[Document]
EmbeddedFiles=2
```

```
EmbeddedFile1=*.pdf
EmbeddedFile2=*.joboptions
```

ProhibitEmbeddedFiles

Key: ProhibitEmbeddedFiles
Error code: CHK_E_EF

Use the option ProhibitEmbeddedFiles to check for embedded files.

true Raise error if document contains embedded files.

false Do not check for embedded files.

Example: Disallow embedded files.

```
[Document]
ProhibitEmbeddedFiles=true
```

5.5.3 [Pages] INI-File Section

PageSizes, PageSize‹i› (Approved page sizes)

Key: PageSizes

Key: PageSize‹i›
Error code: CHK_E_PAGESIZE

Approved page sizes are specified by setting PageSizes=‹n›, where ‹n› is the count of page sizes, i.e. a value larger than 0. Sizes are defined using PageSize‹i›=‹Size i›, where ‹i› is an index ranging from 1 to ‹n› and ‹Size i› is one of the following size specifications:

Letter US Letter page 8.5 x 11 in.

A‹k› A series international paper size standard A0 to A10.

DL DIN long paper size 99 x 210 mm.

‹w› x ‹h› ‹uu› Arbitrary page size of width ‹w›, height ‹h› measured in units ‹uu›. Supported units are in, pt, cm, and mm.

The tolerance used for size comparison is 3 points (3/72 inch, approximately 1 mm), unless the key SizeTolerance (Tolerance for page size comparison) is specified.

Example:

```
[Pages]
PageSizes=4
```

```
PageSize1=A0
PageSize2=A3
PageSize3=15.53 x 15.35 in
PageSize4=181 x 181 mm
```

SizeTolerance (Tolerance for page size comparison)

Key: SizeTolerance

Tolerance used for page size comparison.

Percentage Proportional difference, e.g. SizeTolerance=10%.

Absolute value Absolute difference in points (1/72 inch), e.g. SizeTolerance=72 allows 1 inch.

The tolerance used for size comparison is 3 points (3/72 inch), unless the key SizeTolerance is specified.

Example: Allow a tolerance of 10%.

```
[Pages]
SizeTolerance=10%
```

EmptyPage

Key: EmptyPage
Error code: CHK_E_EMPTYPAGE

Use the key EmptyPage to disallow empty pages. A page is considered empty if no graphic objects are drawn onto it.

true Raise error if the page is not empty.

false Raise error if the page is empty.

Example: Raise error CHK_E_EMPTYPAGE, if the document contains an empty page.

```
[Pages]
EmptyPage=false
```

MaxPageSize

Key: MaxPageSize
Error code: CHK_E_MAXPAGESIZE

Use the key MaxPageSize to disallow pages exceeding the specified size in any dimension. The tolerance for size comparison is specified by the key SizeTolerance. Both portrait and landscape variants of MaxPageSize are allowed.

See PageSize‹i› for a description of supported page size formats.

Example: Raise error CHK_E_MAXPAGESIZE, if the document contains a page larger than A4.

```
[Pages]
MaxPageSize=A4
```

RequirePageResources

Key: RequirePageResources
Error code: CHK_E_PAGERESOURCES

Test if pages contain an explicitly associated resource dictionary.

true Raise error if the page does not have resource dictionary.

Note that it is allowed for a page to not have an explicitly associated resource dictionary if it is inherited from the pages tree. The 3-Heights® PDF Validator API always validates that all pages have a resource dictionary.

Example: Raise error CHK_E_PAGERESOURCES, if the document contains a page without a resource dictionary.

```
[Pages]
RequirePageResources=false
```

5.5.4 [Graphics] INI-File Section

ImageMaxDPI (Maximum resolution of images)

Key: ImageMaxDPI
Error code: CHK_E_IMGMAXDPI

Use ImageMaxDPI to set maximum allowed resolution in DPI (dots per inch) for all images.

Example: Set the maximum allowed resolution to 602 DPI.

```
[Graphics]
ImageMaxDPI=602
```

ImageMinDPI (Minimum resolution of images)

Key: ImageMinDPI
Error code: CHK_E_IMGMINDPI

Use ImageMinDPI to set minimum allowed resolution in DPI (dots per inch) for all images.

Example: Embedded images must have a resolution from 148 to 152 DPI.

```
[Graphics]
ImageMinDPI=148
ImageMaxDPI=152
```

ScanMaxDPI (Maximum resolution of scanned images)

Key: ScanMaxDPI
Error code: CHK_E_SCANMAXDPI

Use ScanMaxDPI to set maximum allowed resolution in DPI (dots per inch) for scanned images. All images that cover a majority of the page are classified as scanned images.

Example: Set the maximum allowed resolution to 602 DPI.

```
[Graphics]
ScanMaxDPI=602
```

ScanMinDPI (Minimum resolution of scanned images)

Key: ScanMinDPI
Error code: CHK_E_SCANMINDPI

Use ScanMinDPI to set minimum allowed resolution in DPI (dots per inch) for scanned images.

Example: Embedded images must have a resolution from 148 to 152 DPI.

```
[Graphics]
ScanMinDPI=148
ScanMaxDPI=152
```

ScanColor (Color for scanned images)

Key: ScanColor
Error code: CHK_E_SCANCLR

If you do not want to allow color scans, use the ScanColor option.

true Raise error if scanned image does not contain color.

false Raise error if scanned image does contain color.

Example: If you want to disallow color scans.

```
[Graphics]
ScanColor=false
```

OCRText

Key: OCRText
Error code: CHK_E_OCRTEXT

Test, if scanned images have OCR text, i.e. if the file is word-searchable.

true Raise error if scanned image has no OCR text (i.e. file is not word-searchable).

false Raise error if scanned image has OCR text (i.e. file is word-searchable).

Example: Raise an error if an image has no OCR text.

```
[Graphics]
OCRText=true
```

ProhibitColor

Key: ProhibitColor
Error code: CHK_E_CLRUSED

If you only want to allow black and white, use the ProhibitColor option.

true Raise error if the page contains color.

false Do not check for color.

Example:

```
[Graphics]
```

```
ProhibitColor=true
```

ProhibitTransparency

Key: ProhibitTransparency
Error code: CHK_E_TRANSPARENCYUSED

Activate the option ProhibitTransparency to check for transparency. Method for determining transparency on a page is described in detail in Annex A of the ISO 19005-2:2011 standard (PDF/A-2).

true Raise error if the page contains transparency.

false Do not check for transparency.

Example:

```
[Graphics]
ProhibitTransparency=true
```

Layers

Key: Layers
Error code: CHK_E_LAYERS

Use the key Layers to disallow layers.

true Raise error if the document contains no layers.

false Raise error if the document contains layers.

Example: Raise error CHK_E_LAYERS if document contains layers.

```
[Graphics]
Layers=false
```

HiddenLayers

Key: HiddenLayers
Error code: CHK_E_HIDDENLAYERS

Use the key HiddenLayers to disallow hidden layers.

true Raise error if the document contains no hidden layers.

false Raise error if the document contains hidden layers.

Example: Raise error CHK_E_HIDDENLAYERS if document contains hidden layers.

```
[Graphics]
HiddenLayers=false
```

IndexedColorSpaceSize

Key: IndexedColorSpaceSize
Error code: PDF_E_ARRAYSIZE

If you want to check if the indexed colorspace lookup tables are not smaller than the size given by the hival entry, use the IndexedColorSpaceSize option.

true Raise error if a lookup table of an indexed colorspace is too small.

false Do not check for sufficient size of lookup table.

Example:

```
[Graphics]
IndexedColorSpaceSize=true
```

5.5.5 [Fonts] INI-File Section

There are two ways of restricting the allowed fonts used in the validated document. Either every font that is approved is explicitly white-listed or every font that is not approved is black-listed. Most appropriately, only one of the two settings is used at once.

Fonts, Font‹i› (Approved Fonts)

Key: Fonts

Key: Font‹i›
Error code: CHK_E_FONT

Restrict the approved fonts to a defined set of fonts. The number of approved fonts is set by Fonts=‹n›, where ‹n› is a number larger than 0. The names of the approved fonts are listed using Font‹i›=‹fontname i›, where ‹i› is an index ranging from 1 to ‹n› and ‹fontname i› is a font name. Wildcards are supported. Font styles are defined by adding a command and the style after the font family name.

Example: A list of approved fonts can be defined like this:

```
[Fonts]
Fonts=163
Font1=AdvC39b
```

```
Font2=AdvC39b
Font3=AdvHC39b
Font4=AdvHC39b
Font5=Arial
Font6=Arial,Bold
...
Font163=ZapfDingbats
```

NonFonts, NonFont‹i› (Non-approved fonts)

Key: NonFonts

Key: NonFont‹i›
Error code: CHK_E_FONT

A list of non-approved fonts can be defined. Wildcards are supported.

Example:

```
[Fonts]
NonFonts=4
NonFont1=MSTT*
NonFont2=T1*
NonFont3=T2*
NonFont4=T3*
```

Subsetting

Key: Subsetting
Error code: CHK_E_FNTSUB

Subsetting a font means only those glyphs that are actually used are embedded in the font program. Subsetting is mainly used to keep the file size small. The setting Subsetting can be used to test the subsetting of embedded fonts.

true Raise error if embedded font is not subset.

false Raise error if embedded font is subset.

Example: Require all fonts to be subset.

```
[Fonts]
Subsetting=true
```

NonStdEmbedded

Key: NonStdEmbedded
Error code: CHK_E_FNTEMB

The NonStdEmbedded setting can be used to test the embedding of non-standard fonts.

true Raise error if non-standard font is not embedded.

false Raise error if non-standard font is embedded.

Example: Require all non-standard fonts to be embedded.

```
[Fonts]
NonStdEmbedded=true
```

Embedding, EmbeddingExcFonts, EmbeddingExcFont‹i› (Embedding of fonts)

Key: Embedding

Key: EmbeddingExcFonts

Key: EmbeddingExcFont‹i›
Error code: CHK_E_FNTEMB

The Embedding setting can be used to test the embedding of fonts that are used for rendering. The EmbeddingExcFonts and EmbeddingExcFont‹i› keys define a list of fonts exempt from the test.

true Raise error if a font is neither embedded nor in the list of exceptions.

false Raise error if a font is embedded and not in the list of exceptions.

Note that this test works independently of NonStdEmbedded.

Example: Require all fonts except “Albertus” and “Courier” to be embedded.

```
[Fonts]
Embedding=true
EmbeddingExcFonts=2
EmbeddingExcFont1=Albertus*
EmbeddingExcFont2=Courier*
```

5.5.6 [Interactive features] INI-File Section

Annotations, Annotation‹i› (Approved annotations)

Key: Annotations

Key: Annotation‹i›
Error code: CHK_E_ANNOTATION

Set a list of approved annotations.

Example: Allow form fields (Widget annotations) and links (Link annotations) only.

```
[Interactive Features]
Annotations=2
Annotation1=Widget
Annotation2=Link
```

NonActions, NonAction‹i› (Non-approved actions)

Key: NonActions

Key: NonAction‹i›
Error code: CHK_E_ACTION

Set a list of non-approved actions.

Example: Disallow URI-actions.

```
[Interactive Features]
NonActions=1
NonAction1=URI
```

5.5.7 [Digital signatures] INI-File Section

Provider

Key: Provider

To use the signature validation feature of the 3-Heights® PDF Validator API, a cryptographic provider is required. The cryptographic provider implements cryptographic algorithms. If signature validation is active but no valid cryptographic provider is configured, SetProfile, SetProfileMem, SetProfileStream failsthe 3-Heights® PDF Validator API does not start validation and aborts with a return code 3.

The following cryptographic providers are supported:

PKCS#11 Provider

The provider configuration string has the following syntax:

Provider=‹PathToDll›;‹SlotId›

- ‹PathToDll› is the path to driver library filename, which is provided by the manufacturer of the HSM, UBS token or smartcard. The bitness of the DLL and the 3-Heights® PDF Validator API must match. For more information and installation instructions, see TechNotePKCS11.pdf.

  Example:
  - openCryptoki soft store on Linux uses libopencryptoki.so

  - PKCS#11 soft-token on Solaris `libpkcs11.so`
- `‹SlotId›` is optional. If it is not defined, it is searched for the first slot that contains a token.

Windows Cryptographic Provider

This provider uses Windows infrastructure to access certificates and to supply cryptographic algorithms. Microsoft Windows offers two different APIs: the Microsoft CryptoAPI and Cryptography API Next Generation (CNG). The latter is used if the operating system is at least Windows Vista or Windows Server 2008.

The provider configuration string has the following syntax:

`Provider=[‹ProviderType›:]‹Provider›`

The `‹ProviderType›` is optional. An empty `‹Provider›` uses the default provider. If CNG is available, `‹ProviderType›` and `‹Provider›` are both optional.

Example:

- `Provider=`
  The default provider is suitable for all systems where CNG is available.
- `Provider=Microsoft Base Cryptographic Provider v1.0`
- `Provider=PROV_RSA_AES:Microsoft Enhanced RSA and AES Cryptographic Provider`
  The Microsoft CryptoAPI provider type `PROV_RSA_AES` supports the SHA-2 hash algorithms for signature validation. This provider type is recommended in order to validate signatures if neither a PKCS#11 device nor CNG are available.

Example: Use openCryptoki to validate signatures. openCryptoki must be installed and the exact location of the PKCS#11 dll depends on your openCryptoki installation.

```
[Digital Signatures]
Provider=/usr/lib64/opencryptoki/libopencryptoki.so
```

Validation of the following signature types is supported:

- `adbe.pkcs7.sha1`
- `adbe.pkcs7.detached`

> **Terminate method:** If signature validation is activated, the `Terminate` method must be called before application shutdown.

`ValidateNewest` (Validate newest signature)

Key: `ValidateNewest`
Error code: `CHK_E_SIGVAL`

Validate the newest signature of the document. Also see the `Provider` and `Criteria, Criterion‹i›` (Signature validation criteria) keys.

Example: Validate the newest signature using openCryptoki.

```
[Digital Signatures]
ValidateNewest=true
Provider=libopencryptoki.so
Criteria=1
Criterion1=Verification
```

Criteria, Criterion‹i› (Signature validation criteria)

Key: Criteria

Key: Criterion‹i›

List of signature validation criteria. Currently supported are:

Verification The signature can be verified, i.e. the cryptographic message syntax (CMS) is correct and the document has not been modified.

EntireDoc Require that the document has not been updated after the newest signature.

Visible Signature must be visible.

Example: see ValidateNewest (Validate newest signature) key.

5.6 Error handling

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

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

Example:

```
public Boolean Open(string file, string password)
{
  if (!validator.Open(file, password))
  {
    if (validator.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}", validator.ErrorCode, validator.ErrorMessage));
      return false;
    }
```

```
  }
  [...]
}
```

6 Interface reference

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

6.1 PDFValidator Interface

6.1.1 Categories

Property (get): `Long Categories`

Instead of a detailed report using `GetFirstError` and `GetNextError`, there is the alternative to report a summary. The summary consists of 19 possible messages (see `CategoryText` property). If any violation is detected at least once, it is reported once. The value of the `Categories` property is accessed and used after the validation. It returns a number in which each bit represents one of these 19 messages. The textual value for each bit can be retrieved using `CategoryText`.

6.1.2 CategoryText

Method: `String CategoryText(TPDFConformanceCategory iCategory)`

Return a textual description for each of the 19 summary messages. The messages are described in `TPDFConformanceCategory`.

Parameter:

`iCategory` [`TPDFConformanceCategory`] The conformance category for which the description is retrieved.

6.1.3 Close

Method: `Boolean Close()`

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

Returns:

`True` The file was closed successfully.

`False` Otherwise.

6.1.4 Compliance

Property (get): `TPDFCompliance Compliance`

This property indicates the conformance used to validate the currently opened document. This is usually the same value as provided in the `Open, OpenMem, OpenStream` method (unless `ePDFUnk` was supplied). This property must be read after `Open, OpenMem, OpenStream`. It is no longer meaningful after a call to `Close`.

6.1.5 ErrorCode

Property (get): `TPDFErrorCode ErrorCode`

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

6.1.6 ErrorMessage

Property (get): `String ErrorMessage`

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

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

.

6.1.7 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.

6.1.8 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.

6.1.9 LicenseIsValid

Property (get): `Boolean LicenseIsValid`
Static

Check if the license is valid.

6.1.10 NoTempFiles

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

If set to `True`, the validator does not create any temporary files. If set to `False`, temporary files may be created, e.g. for embedded files. Use this option with care, because if set to True, this might increase memory consumption significantly.

6.1.11 Open, OpenMem, OpenStream

Method: `Boolean Open(String FileName, String Password, TPDFCompliance Compliance)`

Method: `Boolean OpenMem(Variant MemoryBlock, String Password, TPDFCompliance Compliance)`

Method: `Boolean OpenStream(Variant Stream, String Password, TPDFCompliance Compliance)`

Open a PDF file. 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.

- Path: e.g. `c:\data\document.pdf`
- HTTP URL in the following form:

`http://[‹username›:‹password›@]‹domain›[:‹port›][/‹resource›]`
where `‹username›` and `‹password›` are used for HTTP basic authentication. Example:
`http://myself:secret@site.com:988/documents`
- HTTPS URL: URL beginning with `https://`
- FTP URL: URL beginning with `ftp://`

`Password` [`String`] The user or the owner password of the encrypted PDF document.

`Compliance` [`TPDFCompliance`] The conformance level. See enumeration `TPDFCompliance`. If `ePDFUnk` is passed, the validator determines the claimed conformance of the document. The determined conformance can be read using the `Compliance` property and is used in the `Validate` method. Note that the claimed conformance is not limited to a version of PDF/A.

Returns:

`True` The file could successfully be opened.

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

6.1.12 PageCount

Property (get): `Long PageCount`

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

6.1.13 ProductVersion

Property (get): `String ProductVersion`

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

6.1.14 ReportingLevel

Property (get, set): `Integer ReportingLevel`
Default: `3`

Get or set the reporting level. The supported levels are:

0	None	Nothing is reported
1	Errors	Errors are reported
2	Warnings	Errors and warnings are reported
3	Information	Error, warnings and information are reported

The ReportingLevel property must be set before the Open, OpenMem, OpenStream method to be applied.

6.1.15 SetLicenseKey

Method: Boolean SetLicenseKey(String LicenseKey)

Sets the license key.

6.1.16 SetProfile, SetProfileMem, SetProfileStream

Method: Boolean SetProfile(String FileName)
License feature: Custom

Method: Boolean SetProfileMem(Variant MemoryBlock)
License feature: Custom

Method: Boolean SetProfileStream(Variant Stream)
License feature: Custom

Set custom profile to validate conformance to corporate directives. See Custom validation profiles for more information on features and configuration file format.

Parameter:

FileName [String] The file name of the profile configuration file. Set FileName to Nothing or the empty string to remove the active profile.

Returns:

True Profile was set successfully.

False Error setting profile. See the ErrorCode and ErrorMessage properties for more information on the cause.

6.1.17 StopOnError

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

If set to true, the Validate method aborts on the first validation error; i.e. the validation process stops (error PDF_E_STOPPED) as soon as a problem is found that makes the file non-conforming. This speeds up the validation of non-conforming files.

This property must always be set after Open, OpenMem, OpenStream. It is set to False after a call to Close.

6.1.18 Terminate

Method: Void Terminate()

Terminate all open sessions, and finalize and unload all PKCS#11 drivers. Calling Terminate is mandatory if in the custom validator profile, a PKCS#11 device is configured for signature validation (key Provider). Some drivers require Terminate to be called. Otherwise, your application might crash and/or your HSM, USB token, or smart card may not be unlocked.

Make sure to end all open sessions and dispose of all PDFValidator objects before calling Terminate. After calling Terminate, the process may not call any other methods of this class.

When using the C interface, Terminate may not be called from the context of the destructor of a global or static object, an atexit() handler, nor the DllMain() entry point.

6.1.19 Validate

Method: Boolean Validate()

This method starts the validation. It aborts after the first error if StopOnError is set to true.

Returns:

True The validation finished successfully and the file conforms to the requested standard.

False The validation was aborted (e.g. because an error was found and StopOnError is set to True) or the file does not conform to the requested standard.

The document conforms to the requested standard if Validate returns True.

6.1.20 WriteFontValidationXML

Method: Boolean WriteFontValidationXML(Stream outputStream)

Write font validation information in XML format to a stream. This method must be called after Validate and before Close.

For more information on the structure of the resulting XML, see the XML schema ValidatorFontInformation.xsd and the stylesheet ValidatorFontInformation.xsl in the documentation directory. The latter can be used to view the resulting XML in a web browser if both files are contained in the same directory.

Parameter:

`outputStream` [`Stream`] The stream the font validation information is written to.

Returns:

`True` The font information has been written successfully.

`False` Otherwise.

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 property can be accessed to receive the latest error code. This value should only be read if a function call on the PDF Validator API has returned a value, which signales a failure of the function (see Error handling). See also enumeration `TPDFErrorCode`. Pdftools error codes are listed in the header file `bseerror.h`. Please note that only few of them are relevant for the 3-Heights® PDF Validator 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), PDF as prefix (.NET) or no prefix at all (Java).

6.3.1 TPDFErrorCode Enumeration

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

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

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

TPDFErrorCode table

TPDFErrorCode	Description
PDF_S_SUCCESS	The operation was completed successfully.
LIC_E_NOTSET, LIC_E_NOTFOUND, ...	Various license management related errors.
PDF_E_FILEOPEN	Failed to open the file.
PDF_E_PASSWORD	The authentication failed due to a wrong password.
PDF_E_UNKSECHANDLER	The file uses a proprietary security handler, e.g. for a proprietary digital rights management (DRM) system.

TPDFErrorCode table

PDF_E_XFANEEDSRENDERING	The file contains unrendered XFA form fields, i.e. the file is an XFA and not a PDF file. The XFA (XML Forms Architecture) specification is referenced as an external document to ISO 32'000-1 (PDF 1.7) and has not yet been standardized by ISO. Technically spoken, an XFA form is included as a resource in a shell PDF. The PDF's page content is generated dynamically from the XFA data, which is a complex, non-standardized process. For this reason, XFA is forbidden by the ISO Standards ISO 19'005-2 (PDF/A-2) and ISO 32'000-2 (PDF 2.0) and newer.
PDF_E_INVCOMPLIANCE	Invalid or unsupported PDF conformance specified. Either the parameter's value is invalid or the parameter is ePDFUnk and the file does not specify a valid PDF conformance.
PDF_E_STOPPED	Validation aborted, e.g. because an error was found and StopOnError is set to True or because the input file is corrupt.
PDF_E_CONFORMANCE	The file does not conform to the requested standard.

6.3.2 TPDFCompliance Enumeration

TPDFCompliance table

TPDFCompliance	
ePDF13	PDF version 1.3
ePDF14	PDF version 1.4 (corresponds to Acrobat 5)
ePDF15	PDF version 1.5
ePDF16	PDF version 1.6 (corresponds to Acrobat 7)
ePDF17	PDF version 1.7, ISO 32000-1
ePDF20	PDF version 2.0, ISO 32000-2
ePDFA1a	PDF/A 1a, ISO 19005-1, conformance level A
ePDFA1b	PDF/A 1b, ISO 19005-1, conformance level B
ePDFA2a	PDF/A 2a, ISO 19005-2, conformance level A
ePDFA2b	PDF/A 2b, ISO 19005-2, conformance level B
ePDFA2u	PDF/A 2u, ISO 19005-2, conformance level U
ePDFA3a	PDF/A 3a, ISO 19005-3, conformance level A
ePDFA3b	PDF/A 3b, ISO 19005-3, conformance level B
ePDFA3u	PDF/A 3u, ISO 19005-3, conformance level U

TPDFCompliance table

ePDFUnk	Validate claimed PDF conformance of input file

Note that only the values listed above are supported.

6.3.3 TPDFConformanceCategory Enumeration

TPDFConformanceCategory table

TPDFConformanceCategory	Description
eConfFormat	The file format (header, trailer, objects, XREF, streams) is corrupted.
eConfPDF	The document doesn't conform to the PDF reference (missing required entries, wrong value types, etc.).
eConfEncrypt	The file is encrypted and the password was not provided.
eConfColor	The document contains device-specific color spaces.
eConfRendering	The document contains illegal rendering hints (unknown intents, interpolation, transfer and halftone functions).
eConfAlternate	The document contains alternate information (images).
eConfPostScript	The document contains embedded PostScript code.
eConfExternal	The document contains references to external content (reference XObjects, file attachments, OPI).
eConfFont	The document contains fonts without embedded font programs or encoding information (CMAPs).
eConfUnicode	The document contains fonts without appropriate character to Unicode mapping information (ToUnicode maps).
eConfTransp	The document contains transparency.
eConfAnnot	The document contains unknown annotation types.
eConfMultimedia	The document contains multimedia annotations (sound, movies).
eConfPrint	The document contains hidden, invisible, non-viewable or non-printable annotations.
eConfAppearance	The document contains annotations or form fields with ambiguous or without appropriate appearances.
eConfAction	The document contains actions types other than for navigation (launch, JavaScript, ResetForm, etc.)
eConfMetaData	The document's metadata is either missing or inconsistent or corrupt.

TPDFConformanceCategory table

`eConfStructure`	The document doesn't provide appropriate logical structure information.
`eConfOptional`	The document contains optional content (layers).

7 Coverage

7.1 All PDF versions

7.1.1 Lexical checks

- Structure of tokens such as keywords, names, numbers, strings etc.
- Structure of the cross-reference table
- File positions in the trailer dictionary, cross reference table, etc.
- Whether a referenced object has the correct object and generation number
- Length attribute of stream objects

7.1.2 Syntactic checks

- Structure of dictionaries, arrays, indirect objects, streams, etc.
- Compression errors, e.g. CCITT, JPEG, Flate, etc.
- Errors in embedded font programs
- Errors in ICC color profiles

7.1.3 Semantic checks

- Required entries in dictionaries, e.g. width entry in an image dictionary
- Inherited attributes
- Value of the parent entries in dictionaries, e.g. page objects
- Type of the dictionary entry's value, e.g. integer, string, name
- Whether the object must be indirect or direct, e.g. a page object must be an indirect object
- Order of operators in content streams
- Number of operands of the operators
- Type of operands of the operators
- Value ranges of the operands
- Unknown referenced resources
- Operand stack overflow and underflow
- Inconsistent information, e.g. if an image has a stencil mask and soft mask at the same time
- Conformance to implementation limits defined by the PDF Reference
- Absence of unredered XFA forms

7.2 Checks specific to PDF/A

7.2.1 Lexical checks

- No header offset
- Presence of a "binary" marker

7.2.2 Semantic checks

All Conformance Levels:

- Presence of a unique file identifier
- Presence of document metadata
- Presence of embedded font programs where needed
- Presence of character to glyph mapping (encoding) information for the fonts
- Presence of an output intent if needed
- Absence of encryption
- Absence of LZW and non-standard filters
- Absence of JavaScript
- Absence of unallowed annotations
- Absence of unallowed actions
- Absence of form fields that are generated on the fly
- Absence of embedded PostScript code
- Absence of invisible, hidden or non-printable annotations
- Absence of device-specific color spaces
- Absence of unknown rendering intents
- Absence of image interpolation
- Absence of externally referenced information (external streams, reference XObjects, etc.)
- Absence of Open Print Interface (OPI) information
- Absence of alternate images
- Absence of color transfer and half-toning functions

Additional checks for PDF/A-1

- Absence of JPX
- Absence of layers
- Absence of transparency
- Absence of embedded files
- Absence of XRef streams
- Conformity of metadata

Additional checks for PDF/A-2

- PDF/A conformance of embedded files
- Consistency of spot colors

Additional Checks for Level A and U (PDF/A-1a, PDF/A-2a, PDF/A-2u, PDF/A-3a, PDF/A-3u)

- Presence of Unicode information of fonts where needed

Additional Checks for Level A (PDF/A-1a, PDF/A-2a, PDF/A-3a)

- Presence of logical structure information (tagging)
- Presence of alternate descriptions of content (replacement text) where needed

7.3 Supported PDF versions

The 3-Heights® PDF Validator API currently validates the following versions of the PDF Reference and PDF/A ISO standard:

Supported PDF versions

PDF 1.x	PDF Reference 1.3 - 1.6
PDF 1.7	PDF 1.7, ISO 32000-1
PDF 2.0	PDF 2.0, ISO 32000-2
PDF/A-1a	PDF/A-1a, ISO 19005-1, Level A conformance
PDF/A-1b	PDF/A-1b, ISO 19005-1, Level B conformance
PDF/A-2a	PDF/A-2a, ISO 19005-2, Level A conformance
PDF/A-2b	PDF/A-2b, ISO 19005-2, Level B conformance
PDF/A-2u	PDF/A-2u, ISO 19005-2, Level U conformance
PDF/A-3a	PDF/A-3a, ISO 19005-3, Level A conformance
PDF/A-3b	PDF/A-3b, ISO 19005-3, Level B conformance
PDF/A-3u	PDF/A-3u, ISO 19005-3, Level U conformance

8 Version history

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

8.1 Changes in versions 6.19–6.27

- **New** key `IndexedColorSpaceSize` in section `Graphics` for custom validation profiles to check whether the indexed colorspace lookup tables are not smaller than the size given by the hival entry.
- **Removed** unnecessary error message for invalid unicode values in the CID map in case actual text is present.
- **Update** license agreement to version 2.9

8.2 Changes in versions 6.13–6.18

No functional changes.

8.3 Changes in versions 6.1–6.12

- **Improved** validation of corrupt DCT stream data.
- **Improved** XML output of method `WriteFontValidationXML()` with new attribute `is-symbolic` in the element `<font>` of simple fonts.
- **New** property `PageCount`.
- [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.

8.4 Changes in version 5

- Custom Validation Profiles
  - **New** key `Linearization` in section `File` to check whether files are linearized.
  - **New** keys `ImageMaxDPI` and `ImageMinDPI` in section `Graphics` to validate the resolution of images.
- **New** additional supported operating system: Windows Server 2019.
- [PHP] **New** extension PHP 7.3 (non thread safe) for Linux.

8.5 Changes in version 4.12

- **Introduced** license feature `Custom`.
- Custom Validation Profiles

    - **New** key `MaxPageSize` in section `Pages` to disallow pages exceeding the specified size in any dimension.
    - **New** key `RequirePageResources` in section `Pages` to test if pages contain an explicitly associated resource dictionary.
    - **New** key `Embedding`, `EmbeddingExcFonts`, and `EmbeddingExcFont‹i›` in section `Fonts` to test the embedding of fonts.
- **Changed** validation of certain numbers: Use lax validation according to the PDF Association's TechNote 0010 for certain numbers that have no effect on the visual appearance of the document.
- **Improved** validation performance, e.g. when reporting many errors or analyzing ICC profiles.
- **Improved** detection of corrupt DCT streams that might cause interoperability issues.
- **New** HTTP proxy setting in the GUI license manager.

Interface PDFValidator

- [.NET, C, Java] **New** method `WriteFontValidationXML`: Write font validation information in XML format to a stream.

8.6 Changes in version 4.11

- **New** support for reading PDF 2.0 documents.
- [PHP] **New** Interface for Windows and Linux. Supported versions are PHP 5.6 & 7.0 (Non Thread Safe). The `PdfValidatorAPI` 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.

8.7 Changes in version 4.10

- **Updated** validation according to the PDF Association's TechNote 0010, which describes some peer-reviewed resolutions to a variety of ambiguities of corner cases of the PDF/A specifications.
- **Improved** stricter validation of font files of embedded fonts.
- **Improved** stricter validation of logical structure information (PDF/A level A).
- Digital signatures
    - **Improved** signature validation.
        - More signature formats supported, most notably the new European PAdES norm. The Windows cryptographic provider now supports the same formats as the PKCS#11 provider.
        - Support signature algorithm RSA with SSA-PSS (PKCS#1v2.1).
- **Improved** robustness against corrupt input PDF documents.
- [C] **Clarified** Error handling of `TPdfStreamDescriptor` functions.

8.8 Changes in version 4.9

- **Improved** support for and robustness against corrupt input PDF documents.
- **Improved** repair of embedded font programs that are corrupt.
- **New** support for OpenType font collections in installed font collection.
- [C] **Changed** return value `pfGetLength` of `TPDFStreamDescriptor` to `pos_t`[7].

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

8.9 Changes in version 4.8

- [.NET, C, COM, Java] **New** method `Terminate` to terminate and unload all cryptographic providers.
- [.NET, C, COM, Java] **New** property `ProductVersion` to identify the product version.
- [.NET] **Deprecated** method `GetLicenseIsValid`.
- [.NET] **New** property `LicenseIsValid`.

9 Licensing, copyright, and contact

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

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

Contact

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