User Manual

3-Heights™ PDF Validator API

Version 4.5

PDF/A compliant

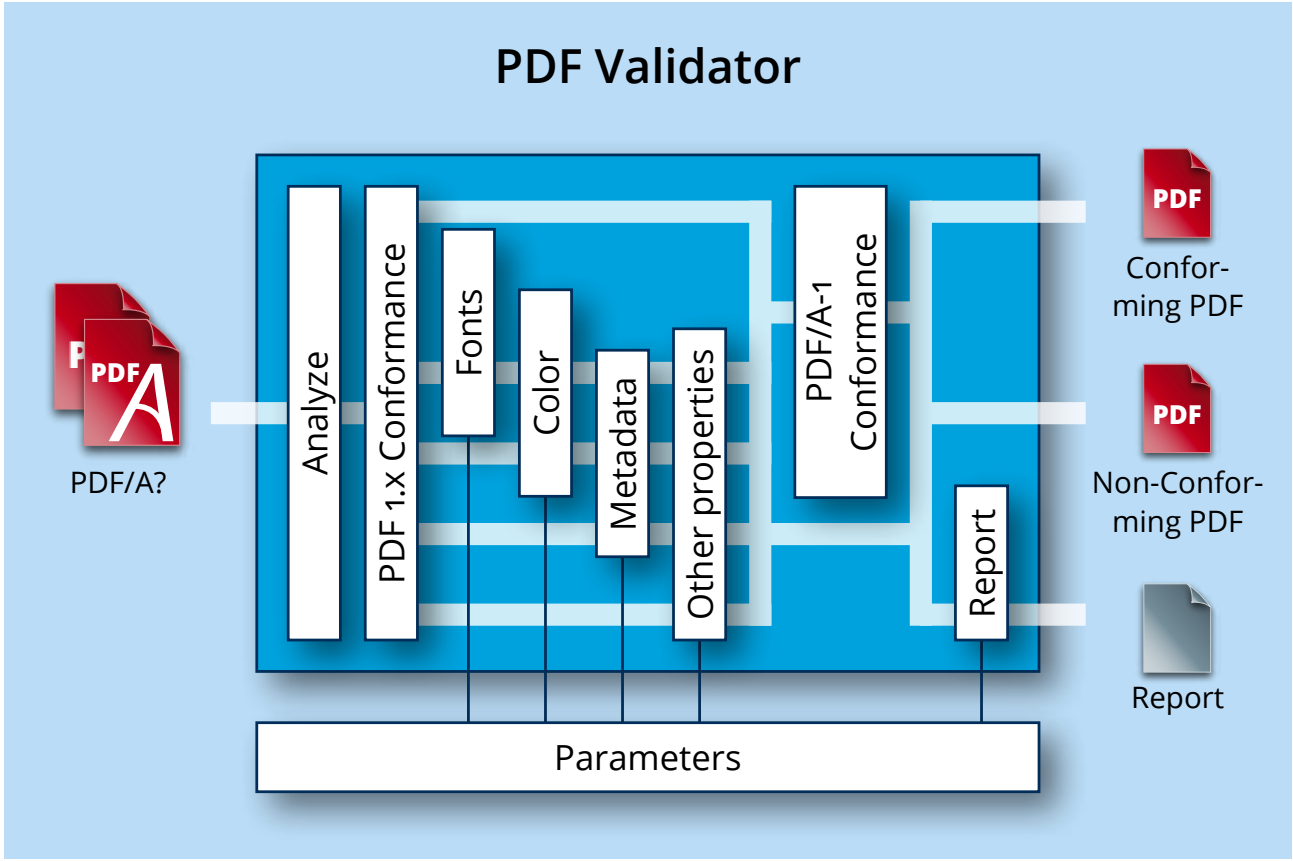
PDF-TOOLS.COM
Premium PDF Technology

Contents

1 Introduction

1.1 Description

The 3-Heights™ PDF Validator API safeguards the quality of PDF documents. It checks PDF files for compliance with 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 compliance with 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 a developer can integrate the 3-Heights™ PDF Validator API in virtually any application

1.2 Functions

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.

General Functions

- Validate PDF documents on the basis of various PDF specifications (PDF 1.4, PDF/A-1, PDF/A-2, PDF/A-3)
- 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
- Read encrypted PDF files
- Determine claimed compliance of document
- Validate compliance with corporate directives defined in custom profile

Validation Functions

See chapter Coverage

Formats

Input Formats

- PDF 1.x (e.g. PDF 1.4, PDF 1.5, etc.)
- PDF/A-1a, PDF/A-1b
- PDF/A-2a, PDF/A-2b, PDF/A-2u
- PDF/A-3a, PDF/A-3b, PDF/A-3u

Compliance

- Standards: ISO 19005-1 (PDF/A-1), ISO 19005-2 (PDF/A-2) , ISO 19005-3 (PDF/A-3), ISO 32000 (PDF 1.7)
- Quality assurance: Isartor test suite
- Bavaria test suite (unofficial) 2

1.3 Interfaces

The following interfaces are available: C, Java, .NET, COM.

1.4 Operating Systems

- Windows XP, Vista, 7, 8, 8.1 - 32 and 64 bit
- Windows Server 2003, 2008, 2008 R2, 2012, 2012 R2 - 32 and 64 bit
- HP-UX 11 and later PA-RISC2.0 32 bit or HP-UX 11i and later ia64 (Itanium) 64 bit
- IBM AIX 5.1 and later (64 bit)
- Linux (32 and 64 bit)
- Mac OS X 10.4 and later (32 and 64 bit)
- Sun Solaris 2.8 and later, SPARC and Intel
- FreeBSD 4.7 and later 32 bit or FreeBSD 9.3 and later 64 bit (on request)

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 many 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 Guide for general information about the API. Read Programmer's Reference for specific information about the functions of the API.

2 Installation and Deployment

2.1 Windows

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

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

   bin: Contains the runtime executable binary code.

   doc: Contains documentation files.

   include: Contains header files to include in your C / C++ project.

   samples: Contains sample programs in various programming languages.

   There is the option to download the software as MSI file, which makes the installation easier.
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 chapter Interfaces.

2.2 Unix

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

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

- Java interface
- Native C interface

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

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

Example code written in different programming languages are available at product page of the PDF Tools AG website (http://www.pdf-tools.com).

All Unix Platforms

1. Unpack the archive in an installation directory, e.g. */opt/pdf-tools.com/*
2. Copy or link the shared object into one of the standard library directories, e.g:
   `ln -s /opt/pdf-tools.com/bin/libPdfValidatorAPI.so /usr/lib`
3. Verify that the GNU shared libraries required by the product are available on your system:
   - *On Linux*: `ldd libPdfValidatorAPI.so`
   - *On AIX*: `dump -H libPdfValidatorAPI.so`

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

MAC OS/X

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

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

2.3 Interfaces

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 examples with which programming languages they can be used.

Table: Interfaces

Interface	Programming Languages
`.NET`	The MS software platform .NET can be used with any .NET capable programming language such as: ▪ C# ▪ VB .NET ▪ J# ▪ others This interface is available in the Windows version only.
`JNI`	The Java native interface (JNI) is for use with Java.
`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++.

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 (e.g. *Inet.dll* is used for http: and other connections. When using the API locally, this file is not used). See also Table: File Description to identify which files are required for your application.

Doc. This file is for documentation only. An empty field indicates this file is not used at all for this particular interface.

Table: Files for Development

Name	.NET	JNI	COM	C
`bin\PdfValidatorAPI.dll`	**Req.**	**Req.**	**Req.**	**Req.**
`bin\pdcjk.dll`	Opt.	Opt.	Opt.	Opt.
`bin\*NET.dll`	**Req.**			
`bin\*NET.xml`	Doc.			
`doc\*.pdf`	Doc.	Doc.	Doc.	Doc.
`doc\PdfValidatorAPI.idl`			Doc.	

doc\javadoc*.*		Doc.		
include\pdfvalidatorapi_c.h				**Req.**
include*.*				Opt.
jar\VALA.jar		**Req.**		
lib\PdfValidatorAPI.lib				**Req.**
samples*.*	Doc.	Doc.	Doc.	Doc.

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

Table: File Description

Name	Description
bin\PdfValidatorAPI.dll	This is the DLL that contains the main functionality (required).
bin\pdcjk.dll	This DLL contains support for Asian languages. It is loaded from the module path.
bin*NET.dll	The .NET assemblies are required when using the .NET interface. The files *bin*NET.xml* contain the corresponding XML documentation for MS Studio.
include\pdferror.h	Supplementary C header file containing error codes.
doc*.*	Various documentations.
include*.*	Contains files to include in your C / C++ project.
jar\VALA.jar	The Java wrapper.
lib\PdfValidatorAPI.lib	The Object File Library needs to be linked to the C/C++ project.
samples*.*	Contains sample programs in different programming languages.

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.

Table: Files for Deployment

Name	.NET	JNI	COM	C
bin\PdfValidatorAPI.dll	**Req.**	**Req.**	**Req.**	**Req.**
bin\pdcjk.dll	Opt.	Opt.	Opt.	Opt.
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 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 .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.
   - Asian text should be supported, thus *pdcjk.dll* is distributed.
3. All file are copied to the target location using a batch script. This script contains the following commands:
   ```
   COPY PdfValidatorAPI.dll %targetlocation%\.
   COPY pdcjk.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 Interface Specific Installation Steps

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 the registration of *PdfValidatorAPI.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\PdfValidatorAPI.dll
```

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`.

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*.

Java Interface

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

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

For execution: Additionally the library *PdfValidatorAPI.dll* needs to be on the Java system property `java.library.path`. This can be achieved by either adding it dynamically at program startup before using the API, or by specifying it using the switch `-Djava.library.path` when starting the Java VM.

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

.NET Interface

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

The .NET assemblies (**NET.dll*) are to be added as references to the project. They are required at compilation time. See also chapter "Getting Started".

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

The native DLL *PdfValidatorAPI.dll* is called by the .NET assembly *PdfValidatorNET.dll*.

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

Alternatively the directory where *PdfValidatorAPI.dll* resides can be added to the environment variable "PATH" or it can simply be copied manually to the output directory.

C Interface

- The header file *pdfvalidatorapi_c.h* needs to be included in the C/C++ program.
- The library *PdfValidatorAPI.lib* needs to be linked to the project.
- The dynamic link library *PdfValidatorAPI.dll* needs to be in path of executables (e.g. on the environment variable "PATH").

2.5 Uninstall, Install a New Version

If you used the MSI for the installation, go to Start ->3-Heights™ PDF Validator API ... ->Uninstall...
If you used the ZIP file: In order to uninstall the product undo all the steps done during installation, e.g. un-register using regsvr32 -u, delete all files, etc.

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

2.6 Note about the Evaluation Version

The evaluation versions of the 3-Heights™ products automatically add a watermark to the output files.

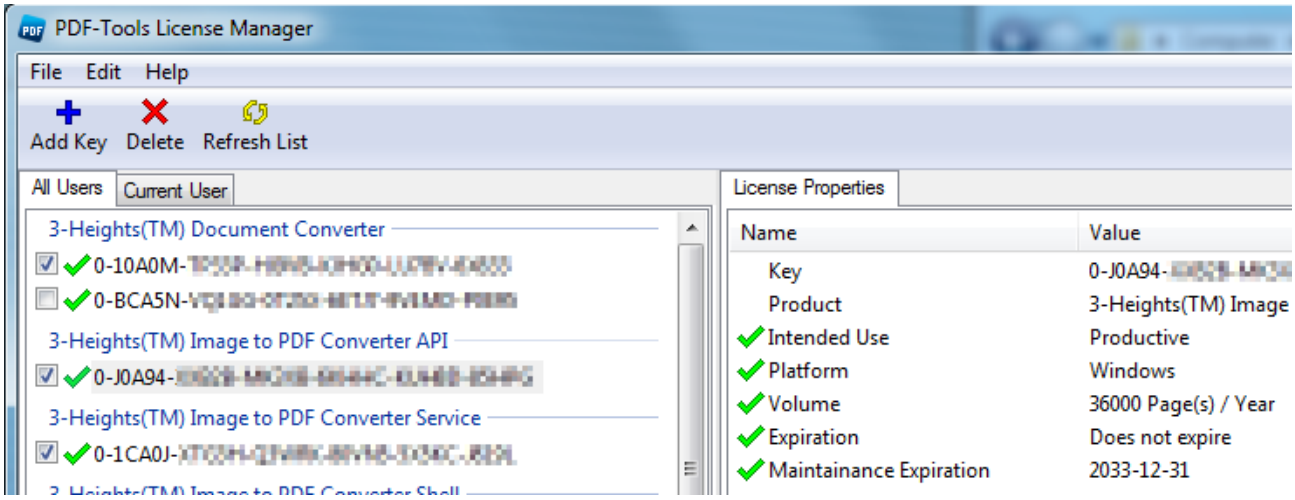
3 License Management

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

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

3.1 Graphical License Manager Tool

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



List all installed license keys

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

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

Add and delete license keys

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

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

Display the properties of a license

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

Select between different license keys for a single product

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

3.2 Command Line License Manager Tool

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

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

```
licmgr
```

List all installed license keys

```
licmgr list
```

Add and delete license keys

Install new license key:

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

Delete old license key:

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

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

- `g`: For all users
- `u`: Current user

Select between different license keys for a single product

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

3.3 License Key Storage

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

Windows

The license keys are stored in the registry:

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

Mac OS X

The license keys are stored in the file system:

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

Unix/Linux

The license keys are stored in the file system:

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

Note: The user, group and permissions of those directories are set explicitly by the license manager tool. It may be necessary to change permissions to make the licenses readable for all users. Example:

```
chmod -R go+rx /etc/opt/pdf-tools
```

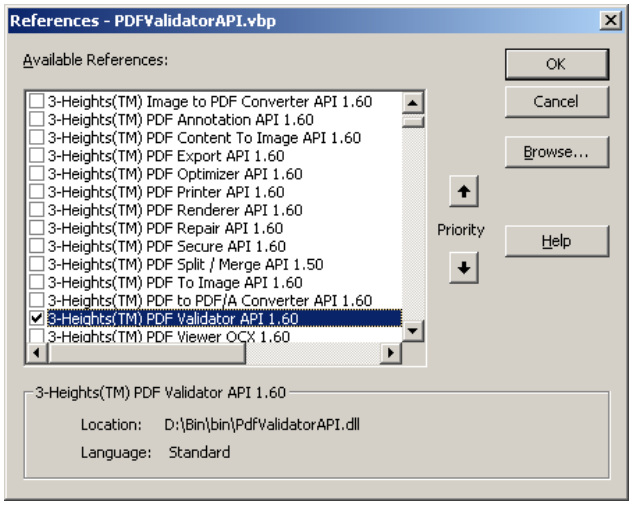
4 Programming Interfaces

4.1 Visual Basic 6

After installing the 3-Heights™ PDF Validator API and registering the COM interface (see chapter Download and Installation), 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, here is a quick start guide for you:

1. First create a new Standard-Exe Visual Basic 6 project. Then include the 3-Heights™ PDF 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
...
```

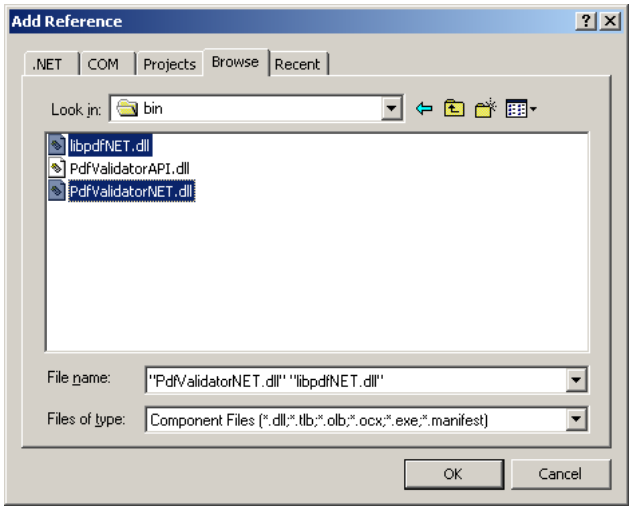
4.2 .NET

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

There should be at least one .NET sample for MS Visual Studio 2005 available in the ZIP archive of the Windows Version of the 3-Heights™ PDF Validator API. The easiest for a quick start is to refer to this sample.

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

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

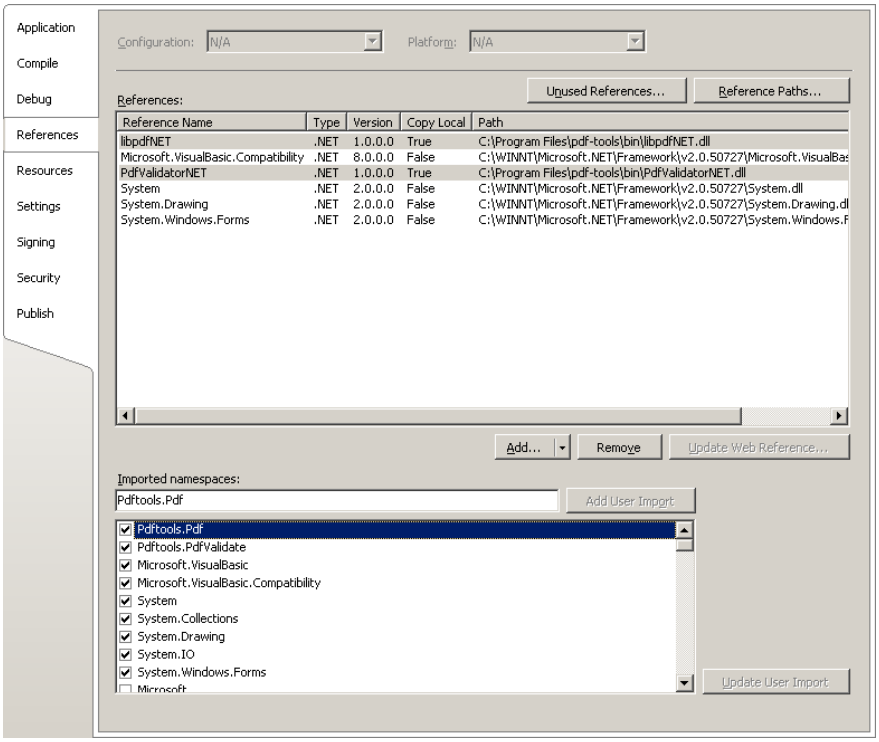


3. import namespaces (Note: This step is optional, but useful.)
4. Write Code

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

Visual Basic

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

```
Dim validator As New PdfValidator
Dim PDFVersion As PDFCompliance = PDFCompliance.ePDFA1b
Dim FileName, Password As String
...
validator.Open(FileName, Password, PDFVersion)
...
```

C#

3. Add the following namespaces:

```
using Pdftools.Pdf;
using Pdftools.PdfValidator;
```

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

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

Deploying in .NET

When deploying a .NET solution, please refer to the following FAQ "Deploying in .NET":
http://www.pdf-tools.com/pdf/Support/FAQ/Article.aspx?name=Deployment-In-NET

Troubleshooting: TypeInitializationException

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

To resolve that ensure the native DLL is found at execution time. For this, see section .NET Interface in the chapter Installation or the following FAQ:
https://www.pdf-tools.com/pdf/Support/FAQ/Article.aspx?name=Exception-type-initializer

4.3 Java

There is a Java sample *validate.java* available which shows how to use the Java interface.

```
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.

```
#include "pdfvalidatorapi_c.h"

TPdfValidator* pDocument;
TPdfValidatorError* pError;

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

5 User's Guide

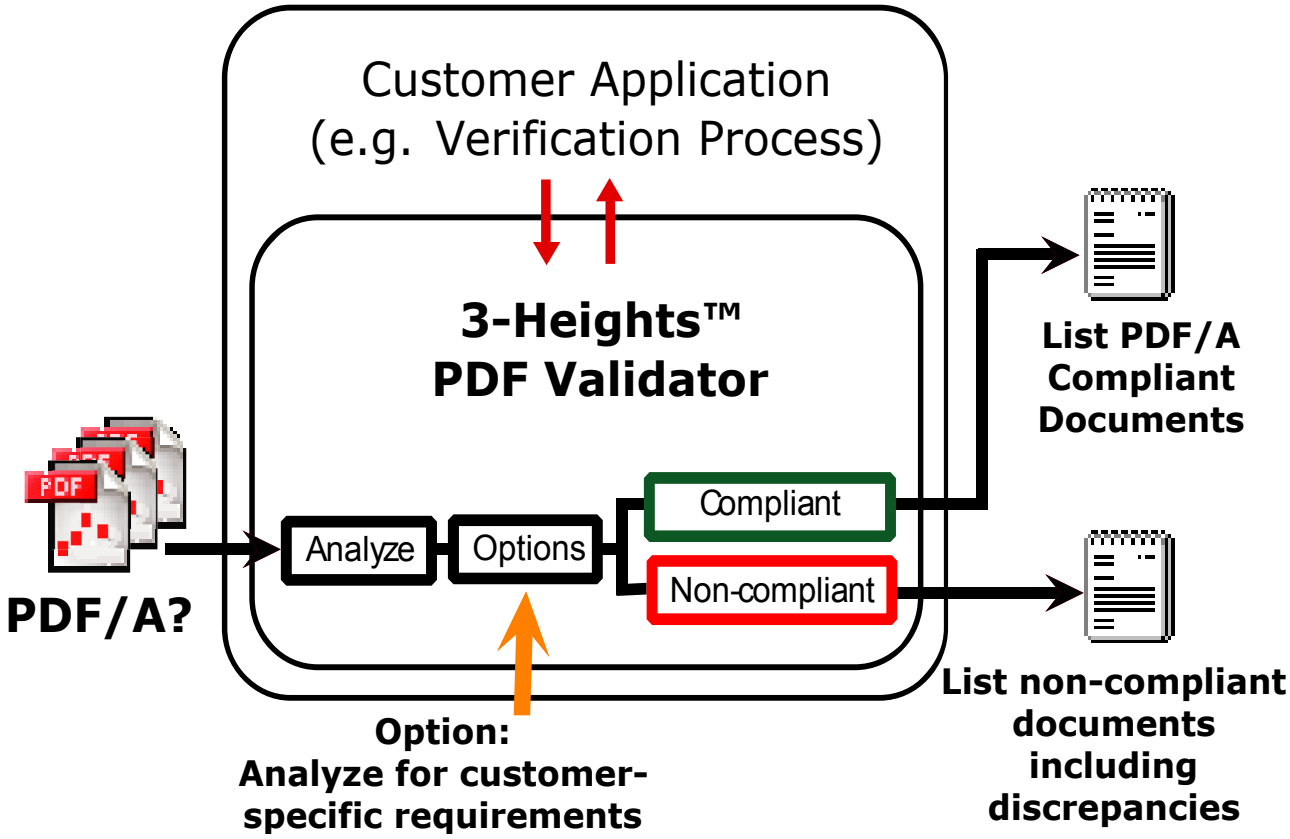
5.1 Overview of the API

What is the 3-Heights™ PDF Validator API about?

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 is compliant or not. If a document is not compliant, it provides detailed information 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 How does the API work in general?

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



1. The API opens a PDF document; at that point a compliance 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:

```
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()
```

Please note that the call sequence above is a bit too simple. If you are using this API for the first time, it might 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:

- 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.
- Instead of listing all errors individually, it is possible to summarize them in 19 generic categories. E.g. if in a PDF document 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 is compliant or not. If additional information is not required, a summary is sufficient.

How to use 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 given range), see additional documentation *vala_custom_extensions_*.pdf*.

5.3 What is PDF/A?

PDF/A-1

PDF/A is an ISO Standard for using PDF format for the long-term archiving of electronic documents. 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-dependant 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

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 shall complement the first part and 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)

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

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.

For additional information about PDF/A please visit: http://www.pdf-tools.com/pdf/pdfa-longterm-archiving-iso-19005-pdf.aspx.

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 PDF/A is described below:

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

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 be compliant with PDF, but not with PDF/A.

A PDF document which 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 PDF/A compliant is still worthwhile.

Error

An error indicates a 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 a PDF to PDF/A converter tool. 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 for compliance with the PDF Reference and PDF ISO standards, the 3-Heights™ PDF Validator API can ensure compliance with custom corporate directives. Custom checks are defined in a configuration file and activated using the SetProfile 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 Section File

File Size Limit 1

```
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
```

File Size Limit 2

```
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, else if he size is larger than FileSize1, CHK_E_FILESIZE1 is raised.

Example: Set allowed file size to 200 MB.

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

Maximum PDF Version

```
Key: MaxPdfVersion
Error Code: CHK_E_MAXPDFVERS
```

The highest PDF version a document may have is defined by the setting MaxPdfVersion. 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
```

Minimum PDF Version

```
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.

Values:

- `true:` Raise error, if file is not encrypted.
- `false:` Raise error, if file is encrypted.

Example: Dis-allow encrypted files.

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

5.5.2 Section Document

Non-Approved PDF Creators

```
Key: NonCreators, NonCreatorX
Error Code: CHK_E_CREATOR
```

Non-approved PDF creators are defined by setting NonCreator='n', where 'n' is the count, i.e. a value larger than 0. Names of the creators are defined using NonCreator1=Name_1 to NonCreator'n'=Name_n.

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

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

Non-Approved PDF Producers

```
Key: NonProducers, 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
```

Allowed Embedded File Types

```
Key: EmbeddedFiles, EmbeddedFileX
Error Code: CHK_E_EFTYPE
```

List of allowed embedded file types. Wild cards 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
```

5.5.3 Section Pages

Approved Page Sizes

```
Key: PageSizes, PageSizeX
Error Code: CHK_E_PAGESIZE
```

Approved page sizes are specified by setting PageSizes='n', where 'n' is the count, i.e. a value larger than 0. Sizes are defined using PageSize1=Size_1 to

Values:

- `Letter:` US Letter page 8.5 x 11 in.
- `A'n':` 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, 1mm), unless the key SizeTolerance is specified.

Example

```
[Pages]
PageSizes=4
PageSize1=A0
PageSize2=A3
PageSize3=15.53 x 15.35 in
PageSize4=181 x 181 mm
```

Tolerance for Page Size Comparison

```
Key: SizeTolerance
Default: 3 (~1mm)
Error Code: n/a
```

Tolerance used for page size comparison.

Values:

- `Percentage:` Proportional difference, e.g. `SizeTolerance=10%`.
- `Absolute Value:` Absolute difference in points (1/72 inch), e.g. "SizeTolenrace=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%
```

Empty Page

```
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.

Values:

- `true:` Raise error, if page is not empty.
- `false:` Raise error, if page is empty.

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

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

5.5.4 Section Graphics

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.

Example: Set the maximum allowed resolution to 602DPI.

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

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
```

Color for Scanned Images

```
Key: ScanColor
Error Code: CHK_E_SCANCLR
```

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

Values:

- `true:` Raise error, if scanned image does not contain color.
- `false:` Raise error, if scanned image does contain color.

Example: If you want to dis-allow color scans.

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

OCR Text

```
Key: OCRText
Error Code: CHK_E_OCRTEXT
```

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

Values:

- `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
```

Prohibit Color

```
Key: ProhibitColor
Error Code: CHK_E_CLRUSED
```

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

Values:

- `true:` Raise error, if page contains color.
- `false:` Do not check for color.

Example

```
[Graphics]
ProhibitColor=true
```

Layers

```
Key: Layers
Error Code: CHK_E_LAYERS
```

Use the key Layers to disallow layers.

Values:

- `true:` Raise error, if document contains no layers.
- `false:` Raise error, if document contains layers.

Example: Raise error CHK_E_LAYERS, if document contains layers.

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

Hidden Layers

```
Key: HiddenLayers
Error Code: CHK_E_HIDDENLAYERS
```

Use the key HiddenLayers to disallow hidden layers.

Values:

- `true:` Raise error, if document contains no hidden layers.
- `false:` Raise error, if document contains hidden layers.

Example: Raise error CHK_E_HIDDENLAYERS, if document contains hidden layers.

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

5.5.5 Section Fonts

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.

Approved Fonts

```
Key: Fonts, FontX
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 Font1=Font_Name_1 to Font'n'=Font_Name_n. Wild cards 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
```

Non-Approved Fonts

```
Key: NonFonts, NonFontX
Error Code: CHK_E_FONT
```

A list of non-approved fonts can be defined, wild cards 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 are embedded in the font program, which are actually used. Subsetting is mainly used to keep the file size small. The setting Subsetting can be used to test the subsetting of embedded fonts.

Values:

- `true:` Raise error, if embedded font is not subsettet.
- `false:` Raise error, if embedded font is subsettet.

Example: Require all fonts to be subsettet.

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

Embedding of Non-Standard Fonts

```
Key: NonStdEmbedded
Error Code: CHK_E_FNTEMB
```

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

Values:

- `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
```

5.5.6 Section Interactive Features

Approved Annotations

```
Key: Annotations, AnnotationX
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
```

Non-Approved Actions

```
Key: NonActions, NonActionX
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 Section Digital Signatures

Provider

```
Key: Provider
Error Code: n/a
```

Set the cryptographic provider used for signature validation.

Example: Use openCryptoki to validate signatures (note that openCryptoki must be installed):

```
[Digital Signatures]
Provider=libopencryptoki.so
```

Validate Newest Signature

```
Key: ValidateNewest
Error Code: CHK_E_SIGVAL
```

Validate the newest signature of the document. Also see the keys Provider and Criteria.

Example: Validate the newest signature using openCryptoki.

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

Signature Validation Criteria

```
Key: Criteria, CriterionX
Error Code: n/a
```

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 key ValidateNewest)

6 Reference Manual

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

6.1 The PDFValidator Interface

Categories

```
Property Long Categories
Accessors: Get
```

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 property CategoryText). If any violation is detected at least once, it is reported once. The value of the property Categories 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(Bit).

CategoryText

```
Property String CategoryText(TPDFConformanceCategory iCategory)
Accessors: Get
```

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

Parameters:
- iCategory: The enumeration of the conformance category.

Close

```
Method Boolean Close()
```

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

Return value:
- True: The file was closed successfully.
- False: Otherwise

Compliance

```
Property TPDFCompliance Compliance
Accessors: Get (after Open)
```

This property indicates the compliance used to validate the currently opened document.

This is usually the same value as provided in the Open method (unless ePDFUnk was supplied).

This property must be read after Open. It is no longer meaningful after a call to Close.

ErrorCode

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

This property can be accessed to receive the latest error code. See also enumeration TPDFErrorCode. PDF-Tools error codes are listed in the header file *pdferror.h*. Please note, that only few of them are relevant for the 3-Heights™ PDF Validator API.

ErrorMessage

```
Property String ErrorMessage
Accessors: Get
```

Return the error message text associated with the last error (see property ErrorCode).

Note, that the property is NULL, if no message is available.

GetFirstError

```
Method PDFError GetFirstError()
```

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

Return value:
- The first error if there are any.
- Nothing otherwise

GetNextError

```
Method PDFError GetNextError()
```

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

Return value:
- The next error if there are any.
- Nothing otherwise

NoTempFiles

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

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

Open

```
Method Boolean Open(String FileName, String Password, TPDFCompliance Compliance)
Method Boolean OpenMem(Variant MemoryBlock, String Password, TPDFCompliance
    Compliance)
Method Boolean OpenStream(Variant StreamDesc, String Password, TPDFCompliance
    Compliance)
```

Open a PDF file. If another document is already open, it is closed first.

Parameters:

- FileName: The file name and optionally the file path, drive or server string according to the operating systems file name specification rules.
  - Path: e.g. `c:\data\document.pdf`
  - HTTP URL: `http://[username:password@]domain[:port][/resource]`, where `username:password` are used for HTTP basic authentication.
  - HTTPS URL: URL beginning with `https://`
  - FTP URL: URL beginning with `ftp://`
- Password: The user or the owner password of the encrypted PDF document.
- Compliance: The compliance level, see enumeration `TPDFCompliance`. If `ePDFUnk` is passed, the validator determines the claimed compliance of the document. The determined compliance can be read using the property Compliance and will be used in the Validate method. Note that the claimed compliance is not limited to a version of PDF/A.

Return value:

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

ReportingLevel

```
Property Integer ReportingLevel
Accessors: Get, Set
Default: 3
```

With this property the reporting level can be set or get. 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 property ReportingLevel must be set before the Open Method in order to be applied.

SetProfile

```
Method Boolean SetProfile(String FileName)
Method Boolean SetProfileMem(Variant MemoryBlock)
Method Boolean SetProfileStream(Variant StreamDesc)
```

Set custom profile to validate compliance with corporate directives. See chapter Custom Validation Profiles for more information on features and configuration file format.

Parameters:
- FileName: The file name of the profile configuration file. Set FileName to null or the empty string, in order to remove the active profile.

Return value:
- True: Profile was set successfully.
- False: Error setting Profile. Consult the properties ErrorCode and ErrorMessage for more information on the cause.

StopOnError

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

If set to true, the method *Validate* will abort on the first error. However, it will continue on warnings and Information messages until the first error occurs.

This property must be set after Open. It is no longer valid after a call to Close.

Validate

```
Method Boolean Validate()
```

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

Return value:
- True: The validation finished successfully
- False: The validation was aborted (e.g. because an error was found and StopOnError is set to true)

The return value does not give an indication whether the document is compliant or not. The document is compliant, if and only if, Validate returns True and the ErrorCode is not set to PDF_E_CONFORMANCE.

6.2 The PDFError Interface

Count

```
Property Long Count
Accessors: Get
```

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

ErrorCode

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

This property can be accessed to receive the latest error code. See also enumeration TPDFErrorCode. PDF-Tools error codes are listed in the header file *pdferror.h*. Please note, that only few of them are relevant for the 3-Heights™ PDF Validator API.

FileName

```
Property String FileName
Accessors: Get
```

Return the name of the (embedded) file in which the error occurred.

Message

```
Property String Message
Accessors: Get
```

This property returns an explaining error message.

ObjectNo

```
Property Long ObjectNo
Accessors: Get
```

This property is not yet supported.

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

PageNo

```
Property Long PageNo
Accessors: Get
```

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).

TPDFErrorCode

All TPDFErrorCode enumerations start with "PDF_" followed by a single letter which is one of "S", "E", "W" or "I", an underscore and a descriptive text. The single letter gives in an indication of the type of error. These are: Success, Error, Warning, Information. With respect to corrupt PDF files: An error indicates a corruption in the PDF, the file may or may not be readable. A warning indicates the file is readable but not valid.

A full list of all PDF Tools error codes is available in the header file *pdferror.h*. Note that only a few are relevant for the PDF Validator API. The most common are listed here.

PDF_S_SUCCESS	The operation was completed successfully.
LIC_E_NOTSET, LIC_E_NOTFOUND, …	Various license management related errors.
PDF_E_FILEOPEN	Failed to open the file.
PDF_E_FILECREATE	Failed to create the file.
PDF_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.

TPDFCompliance

ePDF10	PDF Version 1.0
ePDF11	PDF Version 1.1
ePDF12	PDF Version 1.2
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
ePDFA1a	PDF/A 1a, ISO 19005-1, Level A compliance
ePDFA1b	PDF/A 1b, ISO 19005-1, Level B compliance
ePDFA2a	PDF/A 2a, ISO 19005-2, Level A compliance
ePDFA2b	PDF/A 2b, ISO 19005-2, Level B compliance
ePDFA2u	PDF/A 2u, ISO 19005-2, Level U compliance
ePDFA3a	PDF/A 3a, ISO 19005-3, Level A compliance
ePDFA3b	PDF/A 3b, ISO 19005-3, Level B compliance
ePDFA3u	PDF/A 3u, ISO 19005-3, Level U compliance
ePDFUnk	Unknown format (default)

Note that only ePDF14, ePDFA1a, ePDFA1b, ePDFA2a, ePDFA2b, ePDFA2u, ePDFA3a, ePDFA3b and ePDFA3u are supported.

TPDFConformanceCategory

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 meta data is either missing or inconsistent or corrupt.
`eConfStructure`	The document doesn't provide appropriate logical structure information.
`eConfOptional`	The document contains optional content (layers).

7 Coverage

7.1 All PDF Versions

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

Syntactic Checks

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

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

7.2 Checks Specific for PDF/A

Lexical Checks

- No header offset
- Presence of a “binary” marker

Semantic Checks

All Compliance Levels:

- Presence of a unique file identifier
- Presence of document meta data
- 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 filters
- Absence of Java scripts
- Absence of un-allowed annotations
- Absence of un-allowed 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

Additional Checks for PDF/A-1a, PDF/A-2a, PDF/A-2u, PDF/A-3a, PDF/A-3u

- Presence of Unicode information where needed

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

- Presence of document structure information (tagging)

7.3 Supported PDF Versions

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

`PDF 1.x`	PDF Reference 1.1 - 1.6
`PDF 1.7`	PDF 1.7, ISO 32000-1
`PDF/A-1a`	PDF/A 1a, ISO 19005-1, Level A compliance
`PDF/A-1b`	PDF/A 1b, ISO 19005-1, Level B compliance
`PDF/A-2a`	PDF/A 2a, ISO 19005-2, Level A compliance
`PDF/A-2b`	PDF/A 2b, ISO 19005-2, Level B compliance
`PDF/A-2u`	PDF/A 2u, ISO 19005-2, Level U compliance
`PDF/A-3a`	PDF/A 3a, ISO 19005-3, Level A compliance
`PDF/A-3b`	PDF/A 3b, ISO 19005-3, Level B compliance
`PDF/A-3u`	PDF/A 3u, ISO 19005-3, Level U compliance

8 Licensing and Copyright

9 Contact

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