

User Manual



3-Heights™ PDF to PDF/A Converter API

Version 6.15.0



Contents

1	Introduction	6
1.1	Description	6
1.2	Functions	6
1.2.1	Features	6
1.2.2	Formats	7
	Input Formats	7
	Output Formats	8
1.2.3	Conformance	8
1.3	Interfaces	8
1.4	Operating Systems	8
1.5	How to Best Read this Manual	9
1.6	Digital Signatures	9
1.6.1	Overview	9
1.6.2	Terminology	9
1.6.3	Why Digitally Signing?	10
1.6.4	What is an Electronic Signature?	10
	Simple Electronic Signature	11
	Advanced Electronic Signature	11
	Qualified Electronic Signature	11
1.6.5	How to Create Electronic Signatures	12
	Preparation Steps	12
	Application of the Signature	13
2	Installation and Deployment	15
2.1	Windows	15
2.2	Linux and macOS	16
2.2.1	Linux	16
2.2.2	macOS	17
2.3	Zip Archive	17
2.3.1	Development	18
2.3.2	Deployment	19
2.4	NuGet Package	20
2.5	Interface Specific Installation Steps	20
2.5.1	COM Interface	20
2.5.2	Java Interface	21
2.5.3	.NET Interface	21
2.5.4	C Interface	22
2.6	Uninstall, Install a New Version	22
2.7	Note about the Evaluation License	22
2.8	Special Directories	22
2.8.1	Directory for temporary files	22
2.8.2	Cache Directory	23
2.8.3	Font Directories	23
3	License Management	25
3.1	License Features	25

4	Programming Interfaces	26
4.1	Visual Basic 6	26
4.2	.NET	26
4.2.1	Visual Basic	27
4.2.2	C#	28
4.2.3	Deployment	28
4.2.4	Troubleshooting: TypeInitializationException	28
4.3	ASP	29
5	User's Guide	31
5.1	Process Description	31
5.1.1	Conversion Steps	32
5.1.2	Conversion Errors	32
	Handling Conversion Errors	33
5.1.3	Post Analysis	33
5.2	What is PDF/A?	33
5.2.1	PDF/A-1	34
5.2.2	What is the difference between PDF/A-1b and PDF/A-1a?	34
5.2.3	PDF/A-2	34
5.2.4	PDF/A-3	35
5.3	Color Spaces	35
5.3.1	Colors in PDF	35
	ICC Color Profiles	35
	PDF/A Requirements	36
5.4	Fonts	36
5.4.1	Font Cache	37
5.4.2	Microsoft Core Fonts on Linux or macOS	37
5.4.3	Font Configuration File fonts.ini	37
5.5	Cryptographic Provider	38
5.5.1	PKCS#11 Provider	39
	Configuration	39
	Interoperability Support	39
	Selecting a Certificate for Signing	40
	Using PKCS#11 stores with missing issuer certificates	40
	Cryptographic Suites	40
5.5.2	Windows Cryptographic Provider	41
	Configuration	41
	Selecting a Certificate for Signing	43
	Certificates	43
	Qualified Certificates	45
	Cryptographic Suites	45
5.5.3	SwissSign Digital Signing Service	45
5.5.4	QuoVadis sealsign	47
5.5.5	Swisscom All-in Signing Service	48
	General Properties	48
	Provider Session Properties	49
	On-Demand Certificates	50
	Step-Up Authorization using Mobile-ID	50
5.5.6	GlobalSign Digital Signing Service	50
5.6	How to Create Digital Signatures	52
5.6.1	How to Create a PAdES Signature	53
	Create a PAdES-B-B Signature	54

	Create a PAdES-B-T Signature	54
5.6.2	How to Create a Visual Appearance of a Signature	55
5.6.3	Guidelines for Mass Signing	55
	Keep the session to the security device open for multiple sign operations	55
	Signing concurrently using multiple threads	56
	Thread safety with a PKCS#11 provider	56
5.6.4	Miscellaneous	56
	Caching of CRLs, OCSP, and Time-stamp Responses	56
	How to Use a Proxy	57
	Configuration of Proxy Server and Firewall	57
	Setting the Signature Build Properties	57
5.7	How to Validate Digital Signatures	58
5.7.1	Validation of a Qualified Electronic Signature	58
	Trust Chain	58
	Revocation Information	59
	Time-stamp	59
5.7.2	Validation of a PAdES LTV Signature	60
	Trust Chain	60
	Revocation Information	60
	Time-stamp	61
	LTV Expiration Date	61
	Other PAdES Requirements	61
5.8	Error Handling	61
6	Interface Reference	63
6.1	Pdf2Pdf Interface	63
6.1.1	AddAssociatedFile	63
6.1.2	AddEmbeddedFile	64
6.1.3	AddFontDirectory	64
6.1.4	AddInvoiceXml	64
6.1.5	AddSignature	65
6.1.6	AddZUGFeRDXml	66
6.1.7	AllowDowngrade	66
6.1.8	AllowUpgrade	66
6.1.9	AnalyzeOnly	67
6.1.10	BeginSession	67
6.1.11	ColorSpaceProfile	68
6.1.12	Compliance	68
6.1.13	ConversionErrors	68
6.1.14	ConversionErrorMask	69
6.1.15	Convert, ConvertMem, ConvertStream	69
6.1.16	ConvertAlways	70
6.1.17	EmbedAllFonts	70
6.1.18	EmbedT1asCFF	71
6.1.19	EndSession	71
6.1.20	ErrorCode	71
6.1.21	ErrorMessage	71
6.1.22	ExportText	71
6.1.23	FlattenSignatures	72
6.1.24	ForceEmbeddingOfCMaps	73
6.1.25	GetOCRPuginCount	73
6.1.26	GetOCRPuginName	73

6.1.27	ImageQuality	74
6.1.28	InfoEntry	74
6.1.29	LicenseIsValid	75
6.1.30	Linearize	75
6.1.31	NoCache	76
6.1.32	NoDSS	76
6.1.33	OCRBitonalRecognition	76
6.1.34	OCRDeskewImage	76
6.1.35	OCREmbedBarcodes	77
6.1.36	OCRReembedImages	77
6.1.37	OCRMode	77
6.1.38	OCRResolutionDPI	77
6.1.39	OCRRotatePage	78
6.1.40	OCRThresholdDPI	78
6.1.41	OutputIntentProfile	78
6.1.42	PostAnalyze	78
6.1.43	ProductVersion	79
6.1.44	RemoveSignature	79
6.1.45	ReportDetails	79
6.1.46	ReportSummary	80
6.1.47	SetLicenseKey	80
6.1.48	SetMetadata	80
6.1.49	SetOCREngine	81
6.1.50	SetOCRLanguages	81
6.1.51	SetOCRParams	82
6.1.52	SetSessionProperty	82
6.1.53	SetToUnicodeFile	83
6.1.54	SubsetFonts	84
6.1.55	Terminate	84
6.1.56	TestSession	84
6.1.57	TryConvertEmbPDF	85
6.2	PdfSignature Interface	85
6.2.1	ContactInfo	85
6.2.2	EmbedRevocationInfo	85
6.2.3	FillColor	86
6.2.4	FontName1	86
6.2.5	FontName2	87
6.2.6	Font1Mem	87
6.2.7	Font2Mem	87
6.2.8	FontSize1	87
6.2.9	FontSize2	87
6.2.10	ImageFileName	88
6.2.11	Issuer	88
6.2.12	LineWidth	88
6.2.13	Location	88
6.2.14	Name	88
6.2.15	PageNo	89
6.2.16	Provider	89
6.2.17	Reason	90
6.2.18	Rect	90
6.2.19	SerialNumber	90
6.2.20	SignerFingerprint	90

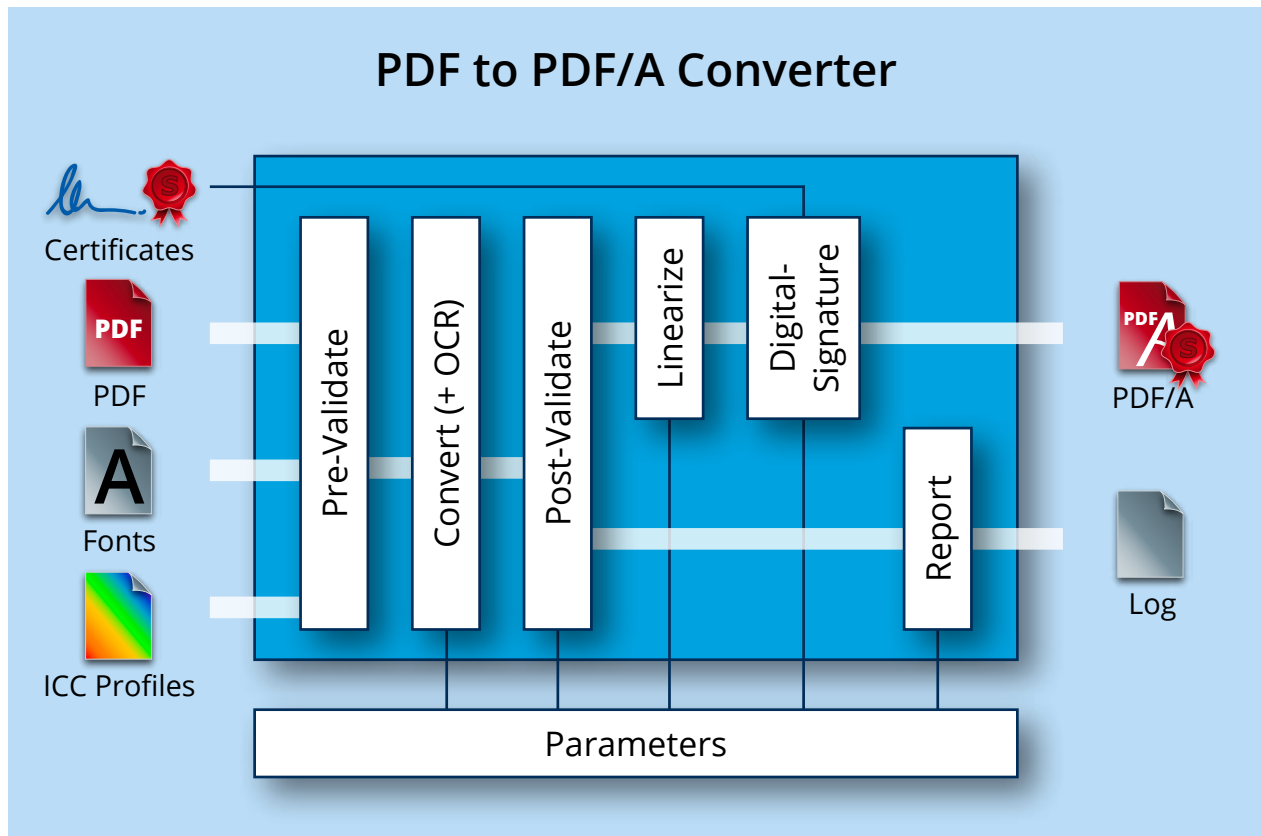
6.2.21	SignerFingerprintStr	91
6.2.22	Store	91
6.2.23	StoreLocation	91
6.2.24	StrokeColor	91
6.2.25	SubFilter	92
6.2.26	Text1	92
6.2.27	Text1Color	92
6.2.28	Text2	93
6.2.29	Text2Color	93
6.2.30	TimeStampCredentials	93
6.2.31	TimeStampURL	93
6.3	Enumerations	94
6.3.1	TPDFCompliance Enumeration	94
6.3.2	TPDFConversionError Enumeration	94
6.3.3	TPDFErrorCode Enumeration	95
6.3.4	TPDFInvoiceType Enumeration	97
	Automatic Profile Detection	97
	ZUGFeRD 1.0 Profiles	98
	ZUGFeRD 2.0 Profiles	98
	ZUGFeRD 2.1 Profiles	98
	Factur-X 1.0 Profiles	98
7	Log File	99
7.1	Warnings and Information	99
7.2	Errors	99
7.3	Reports	99
8	Version History	101
8.1	Changes in Version 6	101
8.2	Changes in Version 5	101
8.3	Changes in Version 4.12	102
8.4	Changes in Version 4.11	102
8.5	Changes in Version 4.10	103
8.6	Changes in Version 4.9	103
8.7	Changes in Version 4.8	104
9	Licensing, Copyright, and Contact	105

1 Introduction

1.1 Description

The 3-Heights™ PDF to PDF/A Converter API converts PDF files into PDF/A files. PDF/A has been acknowledged world-wide as the ISO standard for long-term archiving since 2005. The tool analyzes and converts the input file, applying a digital signature where required.

The integrated validator then optionally checks conformity once again. This product is robust and powerful and therefore predestined for archive migrations of any size.



1.2 Functions

The 3-Heights™ PDF to PDF/A Converter API accepts files from many different applications and automatically converts them into PDF/A. The level of conformity can be set to level A, U, or B. ICC color profiles for device-dependent color profiles and font types are embedded in the document. There is an option to provide the entire character set for fonts (no subsetting) to facilitate editing at a later stage. Missing fonts are reproduced as close to the original as possible via font recognition. Metadata can be generated automatically or added from external sources. The tool also detects and automatically repairs problems typical of the PDF format. A digital signature can be applied and a conformity check carried out at the end of the process. The optional OCR Add-On and linearization for fast web display are valuable additional functions.

1.2.1 Features

- Convert PDF documents to PDF/A-1, PDF/A-2, PDF/A-3

- Support for all PDF/A conformance levels
- Make color spaces device-independent, e.g. by embedding ICC profile or setting an output intent
- Embed and subset fonts
- Colorants management (PDF/A-2 and later)
- Recover corrupt documents
- Repair corrupt data such as embedded font programs or images
- Remove transparency (PDF/A-1 only)
- Remove malicious content such as attached files (PDF/A-1 and PDF/A-2) and JavaScript actions
- Remove multimedia content such as video and sound
- Conversion of embedded and attached files (PDF/A-2 and later)
- Repair metadata and make them consistent
- Conversion process control
 - Pre- and post-validation
 - Conversion reporting
 - Write the application log to a log file
 - Automatically determine optimal conformance based on input file (optional)
 - Enables sophisticated error handling
- Digital signatures, conforming to PDF/A
 - Apply PAdES-LTV (Long Term Validation) signatures
 - Embedded trust chain, time-stamp and revocation information (OCSP, CRL)
 - Various types of cryptographic providers
 - Windows certificate store
 - Hardware such as hardware security module (HSM), smart cards, and USB tokens
 - Online signature services
 - SwissSign Digital Signing Service
 - Swisscom All-in Signing Service
 - GlobalSign Digital Signing Service
 - QuoVadis sealsign
 - Add an optional visual appearance of the signature (page, size, color, position, text, background image, etc.)
 - Mass signing of documents
- Read input from and write output document to file, memory, or stream
- Read encrypted input files
- Enhance output file
 - Set metadata
 - Linearization for fast web view
 - Use PDF file compression features (PDF/A-2 and later)
- Text recognition using OCR engine (optional)
 - Replace old OCR text or skip images with existing OCR text
 - Set the OCR language and options
 - Deskew and de-noise images
 - Detect barcodes
 - List OCR plug-ins
- Add embedded files (PDF/A-2) and associated files (PDF/A-3)
- Embedded XML invoice data conforming to the ZUGFeRD or Factur-X specification (PDF/A-3)

1.2.2 Formats

Input Formats

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

Output Formats

- 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)
 - PAdES (ETSI EN 319 142) signature levels B-B, B-T, CMS
 - Legacy PAdES baseline signature (ETSI TS 103 172) B-Level and T-Level
 - Legacy PAdES (ETSI TS 102 778) Part 2 (PAdES Basic), Part 3 (PAdES-BES), and Part 4 (PAdES-LTV, Long Term Validation)
 - Long term signature profiles for PAdES (ISO 14533-3)
 - Cryptographic Suites (ETSI TS 119 312)
 - ZUGFeRD 1.0, ZUGFeRD 2.0, Factur-X V1.0
- Quality assurance: veraPDF test corpus and Isartor test suite

1.3 Interfaces

The following interfaces are available:

- C
- Java
- .NET Framework
- .NET Core¹
- COM

1.4 Operating Systems

The 3-Heights™ PDF to PDF/A Converter API is available for the following operating systems:

- Windows Client 7+ | x86 and x64
- Windows Server 2008, 2008 R2, 2012, 2012 R2, 2016, 2019 | 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 Limited supported OS versions. [Operating Systems](#)

1.5 How to Best Read this Manual

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

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

1.6 Digital Signatures

1.6.1 Overview

Digital signature is a large and slightly complex topic. This manual gives an introduction to digital signatures and describes how the 3-Heights™ PDF to PDF/A Converter API is used to apply them. It does however not describe all the technical details.

1.6.2 Terminology

Digital Signature is a cryptographic technique of calculating a number (a digital signature) for a message. Creating a digital signature requires a private key from a certificate. Validating a digital signature and its authorship requires a public key. Digital Signature is a technical term.

Electronic Signature is a set of electronic data that is merged or linked to other electronic data in order to authenticate it. Electronic Signatures can be created by means of a digital signature or other techniques. Electronic Signature is a legal term.

Abbreviations

CA	Certification Authority
CMS	Cryptographic Message Syntax
CRL	Certificate Revocation List
CSP	Cryptographic Service Provider
HSM	Hardware Security Module
OCSP	Online Certificate Status Protocol
PKCS	Public Key Cryptography Standards

Abbreviations

QES	Qualified Electronic Signature
TSA	Time-stamp Authority
TSP	Time-stamp Protocol

1.6.3 Why Digitally Signing?

The idea of applying a digital signature in PDF is very similar to a handwritten signature: A person reads a document and signs it with its name. In addition to the name, the signature can contain further optional information, such as the date and location. A valid electronic signature is a section of data that can be used to:

- Ensure the integrity of the document
- Authenticate the signer of the document
- Prove existence of file prior to date (time-stamp)

Digitally signing a document requires a certificate and its private key. How to access and use a certificate is described in the chapter [Cryptographic Provider](#).

In a PDF document, a digital signature consists of two parts:

A PDF related part This part consists of the PDF objects required to embed the signature into the PDF document. This part depends on the signature type (Document Signature, MDP Signature, see table below). Information such as name of the signer, reason, date, location is stored here. The signature may optionally have a visual appearance on a page of the PDF document, which can contain text, graphics and images.

This part of the signature is entirely created by the 3-Heights™ PDF to PDF/A Converter API.

A cryptographic part A digital signature is based on a cryptographic checksum (hash value) calculated from the content of the document that is being signed. If the document is modified at a later time, the computed hash value is no longer correct and the signature becomes invalid, i.e. the validation will fail and will report that the document has been modified since the signature was applied. Only the owner of the certificate and its private key is able to sign the document. However, anybody can verify the signature with the public key contained in the certificate.

This part of the signature requires a cryptographic provider for some cryptographic data and algorithms.

The 3-Heights™ PDF to PDF/A Converter API supports the following types of digital signatures:

Document Signature Check the integrity of the signed part of the document and authenticate the signer's identity. One or more document signatures can be applied. A signed document can be modified and saved by incremental updates. The state of the document can be re-created as it existed at the time of signing.

MDP (Modification detection and prevention) Signature Enable detection of disallowed changes specified by the author. A document can contain only one MDP signature; which must be the first in the document. Other types of signatures may be present.

Document Time-stamp Signature A time-stamp signature provides evidence that the document existed at a specific time and protects the document's integrity. One or more document time-stamp signatures can be applied. A signed document can be modified and saved by incremental updates.

1.6.4 What is an Electronic Signature?

There are different types of electronic signatures, which normally are defined by national laws, and therefore are different for different countries. The type of electronic signatures required in a certain process is usually defined by

national laws. Quite advanced in this manner are German-speaking countries where such laws and an established terminology exist. The English terminology is basically a translation from German.

Three types of electronic signatures are distinguished:

- Simple Electronic Signature “Einfache Elektronische Signatur”
- Advanced Electronic Signature “Fortgeschrittene Elektronische Signatur”
- Qualified Electronic Signature (QES) “Qualifizierte Elektronische Signatur”

All applied digital signatures conform to PDF/A and PAdES.

Simple Electronic Signature

A simple electronic signature requires any certificate that can be used for digital signing. The easiest way to retrieve a certificate, which meets that requirement, is to create a so called self-signed certificate. Self-signed means it is signed by its owner, therefore the issuer of the certificate and the approver of the legitimacy of a document signed by this certificate is the same person.

Example:

Anyone could create a self-signed certificate issued by “Peter Pan” and issued to “Peter Pan”. Using this certificate one is able to sign in the name of “Peter Pan”.

If a PDF document is signed with a simple electronic signature and the document is changed after the signature had been applied, the signature becomes invalid. However, the person who applied the changes, could at the same time (maliciously) also remove the existing simple electronic signature and—after the changes—apply a new, equally looking Simple Electronic Signature and falsify its date. As we can see, a simple electronic signature is neither strong enough to ensure the integrity of the document nor to authenticate the signer.

This drawback can overcome using an advanced or Qualified Electronic Signature.

Advanced Electronic Signature

Requirements for advanced certificates and signatures vary depending on the country where they are issued and used.

An advanced electronic signature is based on an advanced certificate that is issued by a recognized certificate authority (CA) in this country, such as VeriSign, SwissSign, QuoVadis. In order to receive an advanced certificate, its owner must prove its identity, e.g. by physically visiting the CA and presenting its passport. The owner can be an individual or legal person or entity.

An advanced certificate contains the name of the owner, the name of the CA, its period of validity and other information.

The private key of the certificate is protected by a PIN, which is only known to its owner.

This brings the following advantages over a simple electronic signature:

- The signature authenticates the signer.
- The signature ensures the integrity of the signed content.

Qualified Electronic Signature

Requirements for qualified certificates and signatures vary depending on the country where they are issued and used.

A Qualified Electronic Signature is similar to an advanced electronic signature, but has higher requirements. The main differences are:

- It is based on a qualified certificate, which is provided as a hardware token (USB stick, smart card).
- For every signature it is required to enter the PIN code manually. This means that only one signature can be applied at a time.
- Certificate revocation information (OCSP/CRL) can be acquired from an online service. The response (valid, revoked, etc.) must be embedded in the signature.
- A time-stamp (TSP) that is acquired from a trusted time server (TSA) may be required.

This brings the following advantages over an advanced electronic signature:

- The signature ensures the certificate was valid at the time when the document was signed (due to the embedding of the OCSP/CRL response).
- The signature ensures the integrity of the time of signing (due to the embedding of the time-stamp).
- Legal processes that require a QES are supported.

Note: A time-stamp can be added to any type of signature. OCSP/CRL responses are also available for some advanced certificates.

1.6.5 How to Create Electronic Signatures

This is a simple example of how to create an electronic document signature. More detailed examples can be found in [How to Create Digital Signatures](#).

Preparation Steps

1. Identify whether an [Advanced Electronic Signature](#) or a [Qualified Electronic Signature](#) is required. For most automated processes an advanced signature is sufficient.
2. Identify regulatory requirements regarding the content and life cycle of the signature:
 - Is a time-stamp required to prove that the signature itself existed at a certain date and time?
 - Should validation information be embedded, in order to allow the signature to be validated long time after its generation?
 - Should the integrity of the validation material be protected?
 - Is a specific signature encoding required?

These requirements (or regulatory requirements) define the signature level that must be used.

3. Acquire a corresponding certificate from a CA.
For automated processes we recommend to use a HSM, an online signing service, or soft certificates. Other hardware such as USB tokens or Smart Cards are often cheaper, but limited to local interactive single-user applications.

When using an online signing service, ensure that it supports the required signature encoding.

4. Setup and configure the certificate's [Cryptographic Provider](#).
 - In case the certificate resides on hardware such as an USB token or a Smart Card, the required middleware (driver) needs to be installed.
 - In case the certificate is a soft certificate, it must be imported into the certificate store of a cryptographic provider.
5. Optional: Acquire access to a trusted time server (TSA) (preferably from the CA of your signing certificate).
6. Optional: Ensure your input documents conform to the PDF/A standard.
It is recommended to sign PDF/A documents only, because this ensures that the file's visual appearance is well defined, such that it can be reproduced flawlessly and authentically in any environment. Furthermore, PDF/A conformance is typically required if the file is to be archived. Because signed files cannot be converted to PDF/A without breaking its signatures, files must be converted before signing.

Note: A detailed guidance on the use of standards for signature creation can be found in the technical report ETSI TR 119 100.

Application of the Signature

Apply the signature by providing the following information:

1. The [Cryptographic Provider](#) where the certificate is located
2. Values for the selection of the signing certificate (e.g. the name of the certificate)
3. Optional: Time-stamp service URL (e.g. "http://server.mydomain.com:80/tsa")
4. Optional: Time-stamp service credentials (e.g. username:password)
5. Optional: Add validation information
6. Optional: Visual appearance of the signature on a page of the document (e.g. an image).

Example: Steps to Add an Electronic Document Signature

The 3-Heights™ PDF to PDF/A Converter API applies PDF/A conforming signatures. This means if a PDF/A document is digitally signed, it retains PDF/A conformance.

In order to add an electronic document signature with the 3-Heights™ PDF to PDF/A Converter API the following steps need to be done:

1. Create a new [Signature](#) object
2. As value of the [Signature](#)'s name, the name of the certificate that is to be used must be provided. The name of the certificate corresponds to the value "Issued to".
3. If the certificate's private key is PIN protected, the PIN can be passed in the provider configuration.
4. Additional parameters can now be set such as the reason why the signature is applied, etc.

In C# the four steps above look like this:

```
using (Pdf2Pdf converter = new Pdf2Pdf())
{
    using (Signature signature = new Signature())
    {
        signature.Name = "Philip Renggli";
        signature.Provider = "cvp11.dll;0;secret-pin";
        signature.Reason = "I reviewed the document"; // optional
        signature.TimeStampURL = "http://server.mydomain.com:80/tsa"; // optional
        signature.Rect = new PDFRect(10, 10, 210, 60); // optional

        converter.AddSignature(signature);
    }

    if (!converter.Convert("input.pdf", "", "output.pdf", "log.txt"))
        throw new Exception("Unable to convert and sign document: " + doc.ErrorMessage);
}
```

Of course, you would use your own name instead. The name of the certificate is defined by its common name (CN), which is displayed as "issued to" in the Windows Certificate Store.

The visual appearance of the digital signature on a page of the resulting output-document looks as shown below:



Philip Renggli

Digitally signed by

Philip Renggli

Reason: I reviewed the document

Time: D:20061211132331

2 Installation and Deployment

2.1 Windows

The 3-Heights™ PDF to PDF/A Converter API comes as a ZIP archive or as a NuGet package.

The installation of the software requires the following steps.

1. You need administrator rights to install this software.
2. Log in to your download account at <http://www.pdf-tools.com>. Select the product “PDF to PDF/A Converter API”. If you have no active downloads available or cannot log in, please contact pdfsales@pdf-tools.com for assistance.

You will find different versions of the product available. We suggest to download the version, which is selected by default. A different version can be selected using the combo box.

The product comes as a [Zip Archive](#) containing all files, or as a [NuGet Package](#) containing all files for development in .NET.

There is a 32 and a 64-bit version of the product available. While the 32-bit version runs on both, 32 and 64-bit platforms, the 64-bit version runs on 64-bit platforms only. The ZIP archive as well as the NuGet package contain both the 32-bit and the 64-bit version of the product.

3. If you are using the ZIP archive, do the following. Unzip the archive to a local folder, e.g. C:\Program Files\PDF Tools AG\.

This creates the following subdirectories (see also [Zip Archive](#)):

Subdirectory	Description
bin	Contains the runtime executable binaries.
doc	Contains documentation.
include	Contains header files to include in your C/C++ project.
jar	Contains Java archive files for Java components.
lib	Contains the object file library to include in your C/C++ project.
samples	Contains 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 which interface you are using. Perform the specific installation steps for that interface described in [Interface Specific Installation Steps](#).
7. Ensure the cache directory exists as described in chapter [Special Directories](#).
8. Make sure your platform meets the requirements regarding color spaces and fonts described in chapters [Color Spaces](#) and [Fonts](#) respectively.
9. If you want to sign documents, proceed with setting up your cryptographic provider as described in chapter [Cryptographic Provider](#).
10. (Optional) Download and install the 3-Heights™ OCR Enterprise Add-On and the OCR Engine as described in the respective manuals:
 - 3-Heights™ OCR Add-On for ABBYY FineReader Engine v10: [OcrAbbyy10.pdf](#)
 - 3-Heights™ OCR Add-On for ABBYY FineReader Engine v11: [OcrAbbyy11.pdf](#)
 - 3-Heights™ OCR Add-On for ABBYY FineReader Engine v12: [OcrAbbyy12.pdf](#)
 - 3-Heights™ OCR Service: [OcrService.pdf](#) from the separate product kit.

2.2 Linux and macOS

This section describes installation steps required on Linux or macOS.

The Linux and macOS version of the 3-Heights™ PDF to PDF/A Converter API provides two interfaces:

- Java interface
- Native C interface

Here is an overview of the files that come with the 3-Heights™ PDF to PDF/A Converter API:

File Description

Name	Description
bin/x64/libPdf2PdfAPI.so	This is the shared library that contains the main functionality. The file's extension differs on macOS (.dylib instead of .so).
bin/x64/*.ocr	These are OCR plugin modules.
doc/*.*	Documentation
include/*.h	Contains header files to include in your C/C++ project.
jar/CNVA.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 libPdf2PdfAPI.so
```

In case the above reports any missing libraries you have three options:

- a. Download an archive that is linked to a different version of the GNU shared libraries and verify whether they are available on your system. Use any version whose requirements are met. Note that this option is not available for all platforms.
 - b. Use your system's package manager to install the missing libraries. It usually suffices to install the package `libc++6`.
 - c. Use GNU shared libraries provided by PDF Tools AG:
 1. Go to <http://www.pdf-tools.com> and navigate to "Support" → "Utilities".
 2. Download the GNU shared libraries for your platform.
 3. Install the libraries manually according your system's documentation. This typically involves copying them to your library directory, e.g. /usr/lib or /usr/lib64, and running `ldconfig`.
 4. Verify that the GNU shared libraries required by the product are available on your system now.
3. Create a link to the shared library from one of the standard library directories, e.g:

```
ln -s /opt/pdf-tools.com/bin/x64/libPdf2PdfAPI.so /usr/lib
```

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

6. Ensure the cache directory exists as described in chapter [Special Directories](#).
7. Make sure your platform meets the requirements regarding color spaces and fonts described in chapters [Color Spaces](#) and [Fonts](#) respectively.
8. If you want to sign documents, proceed with setting up your cryptographic provider as described in chapter [Cryptographic Provider](#).
9. (Optional) Download and install the 3-Heights™ OCR Enterprise Add-On and the OCR Engine as described in the respective manuals:
 - 3-Heights™ OCR Add-On for ABBYY FineReader Engine v10: [OcrAbbyy10.pdf](#)
 - 3-Heights™ OCR Add-On for ABBYY FineReader Engine v11: [OcrAbbyy11.pdf](#)
 - 3-Heights™ OCR Add-On for ABBYY FineReader Engine v12: [OcrAbbyy12.pdf](#)
 - 3-Heights™ OCR Service: [OcrService.pdf](#) from the separate product kit.

2.2.2 macOS

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

```
ln libPdf2PdfAPI.dylib libPdf2PdfAPI.jnilib
```

2.3 Zip Archive

The 3-Heights™ PDF to PDF/A Converter 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.

Interface	Programming Languages
.NET	<p>The MS software platform .NET can be used with any .NET capable programming language such as:</p> <ul style="list-style-type: none">■ C#■ VB .NET■ J#■ others <p>For a convenient way to use this interface, see NuGet Package.</p>
Java	<p>The Java interface is available on all platforms.</p>
COM	<p>The component object model (COM) interface can be used with any COM-capable programming language, such as:</p> <ul style="list-style-type: none">■ MS Visual Basic■ MS Office Products such as Access or Excel (VBA)■ C++■ VBScript■ others <p>This interface is available in the Windows version only.</p>
C	<p>The native C interface is for use with C and C++. This interface is available on all platforms.</p>

2.3.1 Development

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

Req. This file is required for this interface.

Opt. This file is optional. See also table [File Description](#) to identify which files are required for your application.

Doc. This file is for documentation only.

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

Files for Development

Name	.NET	Java	COM	C
bin\<platform>\Pdf2PdfAPI.dll	Req.	Req.	Req.	Req.
bin*NET.dll	Req.			
bin*NET.xml	Doc.			
bin\<platform>*.ocr	Opt.	Opt.	Opt.	Opt.
doc*.pdf	Doc.	Doc.	Doc.	Doc.
doc\Pdf2PdfAPI.idl			Doc.	
doc\javadoc*.*		Doc.		
include\pdf2pdfapi_c.h				Req.
include*.*				Opt.
jar\CNVA.jar		Req.		
lib\<platform>\Pdf2PdfAPI.lib				Req. ²
samples*.*	Doc.	Doc.	Doc.	Doc.

The purpose of the most important distributed files of is described in table [File Description](#).

File Description

Name	Description
bin\<platform>\Pdf2PdfAPI.dll	This is the DLL that contains the main functionality (required), where <platform> is either Win32 or x64 for the 32-bit or the 64-bit library respectively.

² Not required for Linux or macOS.

File Description

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 Visual Studio.
bin\<platform>*.ocr	These are OCR plugin DLLs that are used in combination with the 3-Heights™ OCR Enterprise Add-On which can be purchased as a separate product. ³
doc*.*	Various documentations.
include*.*	Contains files to include in your C / C++ project.
lib\<platform>\Pdf2PdfAPI.lib	On Windows operating systems, the object file library needs to be linked to the C/C++ project.
jar\CNVA.jar	The Java API archive.
samples*.*	Contains sample programs in different programming languages.

2.3.2 Deployment

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

Files for Deployment

Name	.NET	Java	COM	C
bin\<platform>\Pdf2PdfAPI.dll	Req.	Req.	Req.	Req.
bin*NET.dll	Req.			
bin\<platform>*.ocr	Opt.	Opt.	Opt.	Opt.
jar\CNVA.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 `convert.exe`. Color profiles are not used.
2. The application uses the COM interface and is distributed on Windows only.
 - The main DLL Pdf2PdfAPI.dll must be distributed.
 - All documents used by the application have their fonts embedded (e.g. because they conform to PDF/A), therefore the font related files are not distributed.

³ These files must reside in the same directory as Pdf2PdfAPI.dll.

3. All files are copied to the target location using a batch script. This script contains the following commands:

```
copy convert.exe %targetlocation%\.  
copy Pdf2PdfAPI.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%\Pdf2PdfAPI.dll.
```

2.4 NuGet Package

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

Installation Download the package PdfTools.Pdf2Pdf.6.15.0.nupkg from your account on <https://www.pdf-tools.com/> to some suitable location.

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

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

Development The package PdfTools.Pdf2Pdf.6.15.0.nupkg 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".

In order to use the software, you must first install a license key for the 3-Heights™ PDF to PDF/A Converter 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 to PDF/A Converter 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 Pdf2PdfAPI.dll. Note that in Windows Vista and later, the command needs to be executed from an administrator shell.

```
regsvr32 "C:\Program Files\PDF Tools AG\bin\<platform>\Pdf2PdfAPI.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 to PDF/A Converter 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 `Pdf2PdfAPI.dll`.

2.5.2 Java Interface

The 3-Heights™ PDF to PDF/A Converter API requires Java version 7 or higher.

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

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

For execution Additionally the library `Pdf2PdfAPI.dll` needs to be in one of the system's library directories⁵ or added to the Java system property `java.library.path`. This can be achieved by either adding it dynamically at program startup before using the API, or by specifying it using the switch `-Djava.library.path` when starting the Java VM. Choose the correct subdirectory (`x64` or `Win32` on Windows) depending on the platform of the Java VM⁶.

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

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

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

2.5.3 .NET Interface

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

It is recommended to use the [NuGet Package](#). This ensures the correct handling of both the .NET assemblies and the native library.

Alternatively, the files in the [Zip Archive](#) can be used directly in a Visual Studio project targeting .NET Framework 2.0 or later. To achieve this, proceed as follows.

⁴ Otherwise you get the following message: `LoadLibrary("Pdf2PdfAPI.dll") failed - The specified module could not be found.`

⁵ On Windows defined by the environment variable `PATH` and e.g. on Linux defined by `LD_LIBRARY_PATH`.

⁶ If the wrong data model is used, there is an error message similar to this: `"Can't load IA 32-bit .dll on a AMD 64-bit platform"`

The .NET assemblies (*NET.dll) are to be added as references to the project; They are needed at compile time. Pdf2PdfAPI.dll is not a .NET assembly, but a native library. It is not to be added as a reference to the project. Instead, it is loaded during execution of the application.

For the operating system to find and successfully load the native library Pdf2PdfAPI.dll, it must match the executing application's bitness (32-bit versus 64-bit) and it must reside in either of the following directories:

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

In Visual Studio, when using the platforms "x86" or "x64", the above can be achieved by adding the 32-bit or 64-bit Pdf2PdfAPI.dll respectively as an "existing item" to the project, and setting its property "Copy to output directory" to true. When using the "AnyCPU" platform, then you have to make sure by some other means that both the 32-bit and the 64-bit Pdf2PdfAPI.dll are copied to subdirectories win-x86 and win-x64 of the output directory respectively.

2.5.4 C Interface

- The header file pdf2pdfapi_c.h needs to be included in the C/C++ program.
- On Windows operating systems, the library Pdf2PdfAPI.lib needs to be linked to the project.
- The dynamic link library Pdf2PdfAPI.dll needs to be in a path of executables (e.g. on the environment variable %PATH%).

2.6 Uninstall, Install a New Version

If you have used the ZIP file for the installation: In order to uninstall the product, undo all the steps done during installation, e.g. un-register using regsvr32.exe /u, delete all files, etc.

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

2.7 Note about the Evaluation License

With the evaluation license the 3-Heights™ PDF to PDF/A Converter API automatically adds a watermark to the output files.

2.8 Special Directories

2.8.1 Directory for temporary files

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

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

Windows

1. The path specified by the %TMP% environment variable.
2. The path specified by the %TEMP% environment variable.

3. The path specified by the %USERPROFILE% environment variable.
4. The Windows directory.

Linux and macOS

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

2.8.2 Cache Directory

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

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

Windows

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

Linux and macOS

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

where <TempDirectory> refers to the [Directory for temporary files](#).

2.8.3 Font Directories

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

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

Windows

1. %SystemRoot%\Fonts
2. User fonts listed in the registry key \HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Fonts. This includes user specific fonts from C:\Users\<user>\AppData\Local\Microsoft\Windows\Fonts and app specific fonts from C:\Program Files\WindowsApps
3. directory Fonts, which must be a direct sub-directory of where Pdf2PdfAPI.dll resides.

macOS

1. /System/Library/Fonts
2. /Library/Fonts

Linux

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

3 License Management

The 3-Heights™ PDF to PDF/A Converter API requires a valid license in order to run correctly. If no license key is set or the license is not valid, then most of the interface elements documented in [Interface Reference](#) will fail with an error code and error message indicating the reason.

More information about license management is available in the [license key technote](#).

3.1 License Features

The functionality of the 3-Heights™ PDF to PDF/A Converter API contains one area to which the following license feature is assigned:

Signature Signature creation.

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.

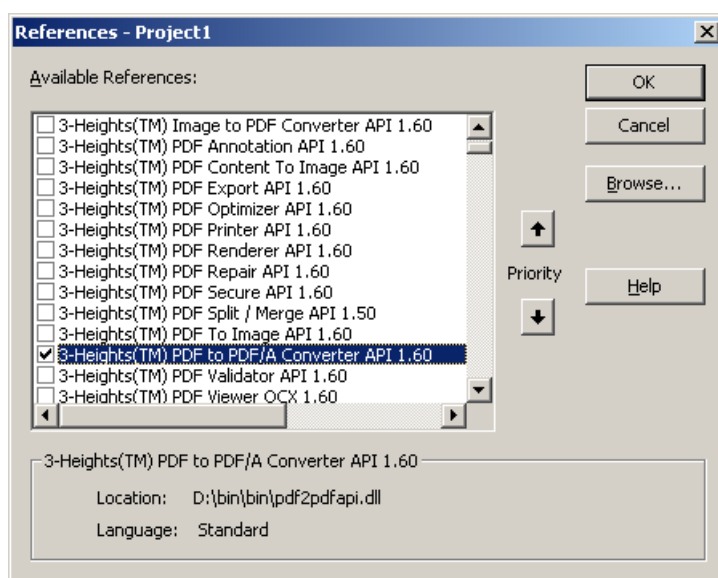
4 Programming Interfaces

4.1 Visual Basic 6

After installing the 3-Heights™ PDF to PDF/A Converter API and registering the COM interface (see [Windows](#)), you find a Visual Basic 6 example with file extension .vpb 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:

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

```
Private Sub Command1_Click()  
    Dim conv As New Pdf2PdfAPI.Pdf2Pdf  
    Dim done As Boolean  
    conv.Compliance = ePDFA1b  
    done = conv.Convert("C:\in1.pdf", "", "C:\out1.pdf", "C:\temp\log1.txt")  
    Set conv = Nothing  
End Sub
```

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 to PDF/A Converter API. The easiest for a quick start is to refer to this sample.

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

1. Start Visual Studio and create a new C# or VB project.
2. Add references to the .NET assemblies.

To do so, in the "Solution Explorer" right-click your project and select "Add Reference...". The "Add Reference" dialog will appear. In the tab "Browse", browse for the .NET assemblies `libpdfNET.dll` and `Pdf2PdfNET.dll`.

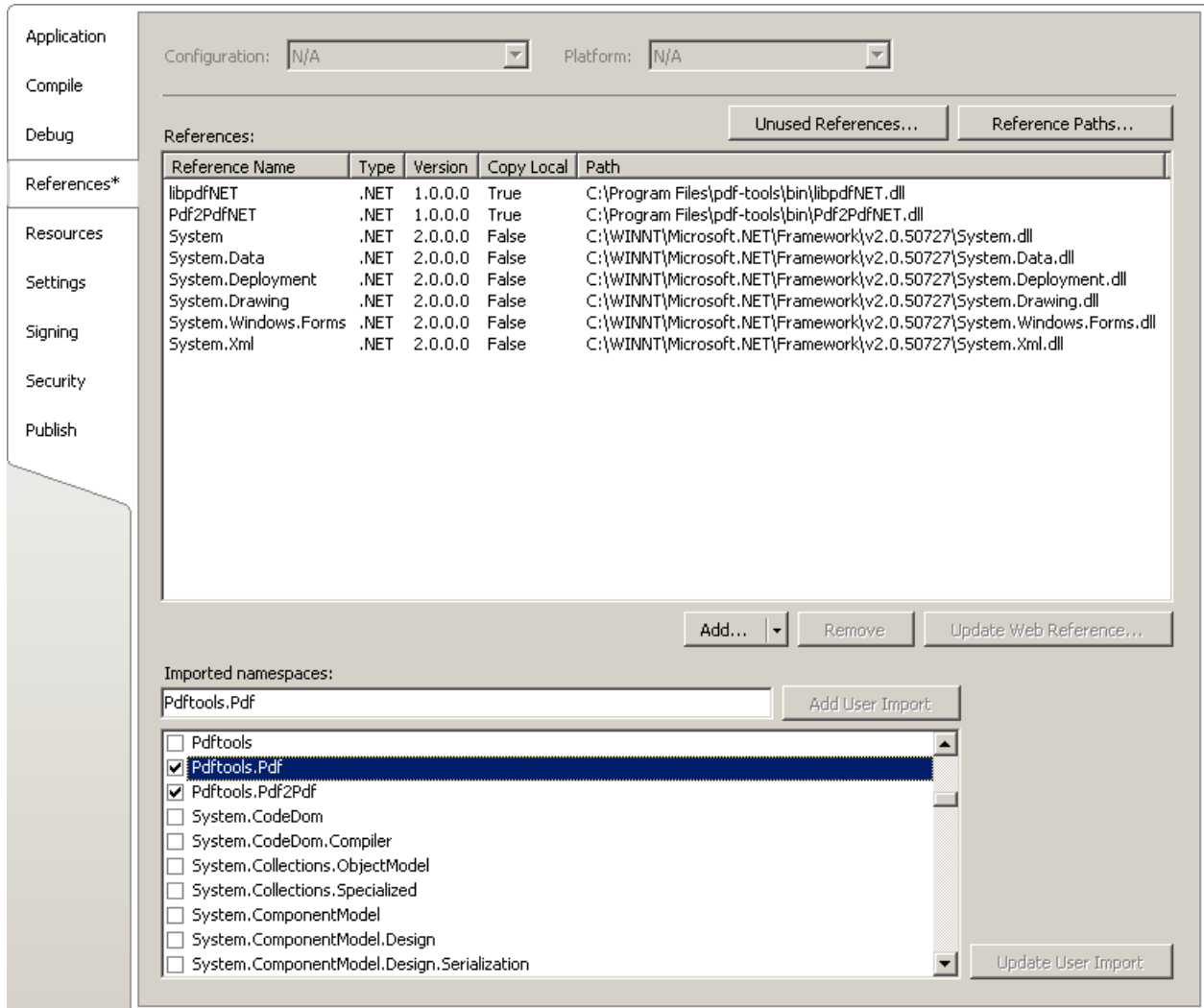
3. Import namespaces (Note: This step is optional, but useful.)
4. Write your code.

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.Pdf2Pdf](#).

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



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

Example:

```
Dim converter As New PdfTools.Pdf2Pdf.Pdf2Pdf()
converter.Compliance = PDFCompliance.ePDFA1b
...
converter.Convert(...)
```

4.2.2 C#

3. Add the following namespaces:

Example:

```
using PdfTools.Pdf;  
using PdfTools.Pdf2Pdf;
```

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

Example:

```
using (Pdf2Pdf converter = new Pdf2Pdf())  
{  
    converter.Compliance = PDFCompliance.ePDFA1b;  
    ...  
    converter.Convert(...);  
}
```

4.2.3 Deployment

This is a guideline on how to distribute a .NET project that uses the 3-Heights™ PDF to PDF/A Converter 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 to PDF/A Converter API's .NET assemblies (*.NET.dll) as well as the native library (Pdf2PdfAPI.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 Pdf2PdfAPI.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 Pdf2PdfAPI.dll is not found at execution time. This normally manifests when the constructor is called for the first time and an exception of type [System.TypeInitializationException](#) is thrown.

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

System.DllNotFoundException Unable to load DLL Pdf2PdfAPI.dll: The specified module could not be found.

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

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

Troubleshooting: DllNotFoundException

This means, that the native DLL Pdf2PdfAPI.dll could not be found at execution time.

Resolve this by either:

- using the [NuGet Package](#).
- adding Pdf2PdfAPI.dll as an existing item to your project and set its property "Copy to output directory" to "Copy if newer", or
- adding the directory where Pdf2PdfAPI.dll resides to the environment variable %Path%, or
- manually copying Pdf2PdfAPI.dll to the output directory of your project.

Troubleshooting: BadImageFormatException

The exception means, that the native DLL Pdf2PdfAPI.dll has the wrong "bitness" (i.e. platform 32 vs. 64 bit). There are two versions of Pdf2PdfAPI.dll available in the [Zip Archive](#): one is 32-bit (directory bin\Win32) and the other 64-bit (directory bin\x64). It is crucial, that the platform of the native DLL matches the platform of the application's process.

(Using the [NuGet Package](#) normally ensures that the matching native DLL is loaded at execution time.)

The platform of the application's process is defined by the project's platform configuration for which there are 3 possibilities:

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

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

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

4.3 ASP

The COM name of the class, for example used in ASP, of the PDF to PDF/A Converter API is:

PDF2PDFAPI.Pdf2Pdf.

```
<%@ Language=VBScript %>
<%
    option explicit
    dim conv
    dim fileNameIn, fileNameOut, logName

    set conv = Server.CreateObject("PDF2PDFAPI.Pdf2Pdf")
    fileNameIn = "C:\PDF-Tools\doc\license.pdf"
    fileNameOut = "C:\temp\output.pdf"
    logName = "C:\temp\output.log"

    conv.ReportSummary = True
    if not conv.Convert(fileNameIn, "", fileNameOut, logName) then
```

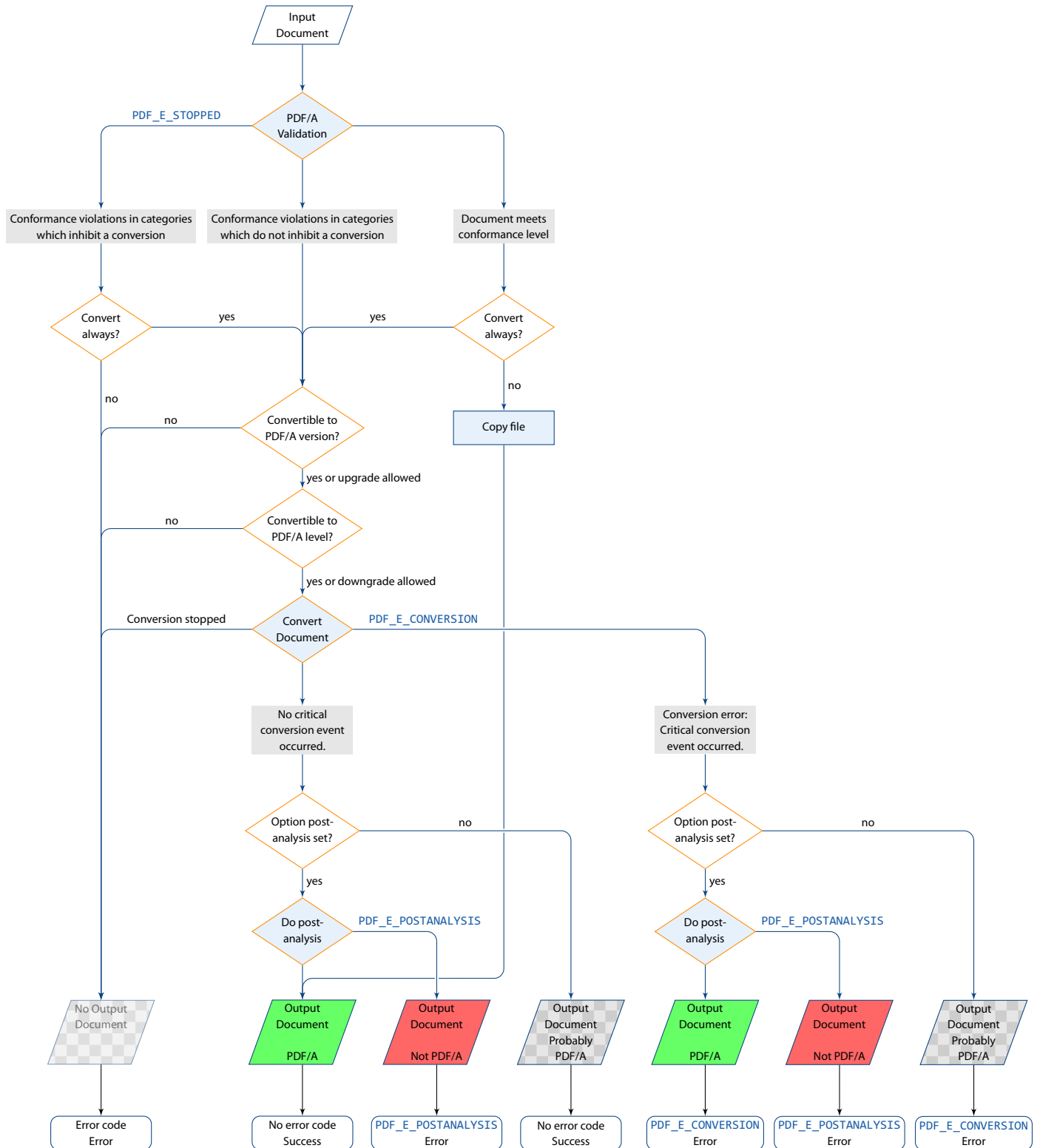
```
        Response.Write "<p>"
        Response.Write "Error converting file: " & conv.ErrorMessage & " <br>"
    else
        Response.Write "<p>"
        Response.Write "Output file created successfully. <br>"
    end if

    Response.Write "<p>"
    Response.Write "Output File (PDF/A-1b) : <a href=" & fileNameOut &
        ">" & fileNameOut & "</a><br>"
    Response.Write "Log File : <a href=" & logName & ">" & logName &
        "</a><br>"
%>
```

5 User's Guide

5.1 Process Description

The workflow of the PDF to PDF/A Conversion is outlined in the graphic below.



- License Check:** The license is checked.
- Pre-Analysis:** The input document is analyzed. If the document already conforms to the requested standard it

is copied.

If the required PDF/A level (e.g. level U or A) cannot be met, the conversion is aborted with an error⁷.

If the target standard is PDF/A-1 and a file contains transparency or other elements that cannot be converted to PDF/A-1, the target standard is upgraded to PDF/A-2 if the property [AllowUpgrade](#) is set to **True**.

If the input document contains non-convertible elements the conversion is stopped, except if convert-always is enabled.

3. **Conversion:** The actual conversion is performed.

The conversion is stopped, e.g. if an OCR error occurs, a required font is not found in the installed font directories, or linearization fails. In this case, no meaningful output document is created.

If actions had to be taken that might have altered the visual appearance of the file or crucial data had to be removed, a conversion error is generated (see chapter [Conversion Errors](#) below).

4. **Post-Analysis:** Finally, the resulting PDF document is validated⁸. If the resulting document does not meet the requested standard a post-analysis error is raised.

5.1.1 Conversion Steps

The goal of the conversion is to create a document which is conforming to the PDF/A ISO standard.

If the analysis of the document indicates a conversion to the requested standard is possible, the following steps are performed:

- Embed and subset non-embedded font programs
- Replace device specific color spaces with CIE-based color spaces
- Add a GTS_PDF/A output intent
- Remove prohibited entries
- Remove entries with a default value
- Remove entries with unknown values
- Add mandatory entries
- Add XMP metadata if missing or fix inconsistent XMP metadata
- Apply implicit optimization functions (e.g. replace and subset embedded fonts)
- Apply implicit repair functions (to conform with ISO19005-1 chapter 6.1)

If the analysis indicates a conversion is not possible, a “best effort” conversion can be forced. In this case the output may or may not be PDF/A conformant. Use the post analysis feature in order to detect, whether or not the output is conformant. It is also possible that the output file looks visually different to the input file due to the forced conversion.

5.1.2 Conversion Errors

The conversion error ([ErrorCode](#) is `PDF_E_CONVERSION` see [TPDFErrorCode](#)) indicates that during conversion, actions had to be taken that might have altered the visual appearance of the file or crucial data had to be removed.

Note: The resulting document conforms to PDF/A nonetheless.

The following issues may result in a conversion error:

- Optional content (layers) removed (PDF/A-1 only)
- Prohibited annotation type converted to stamp
- Prohibited action removed
- Embedded files removed

⁷ Automatic downgrades can be deactivated using the property [AllowDowngrade](#).

⁸ Post-analysis can be deactivated using the property [PostAnalyze](#)

- Transparency removed (PDF/A-1 only)
- Character from show string removed because glyph missing in font
- Unconvertible metadata

All conversion events are written to the log file. The description and location allow to identify potential problems quickly.

For a complete list of conversion events that can lead to a conversion error see the enumeration [TPDFConversionError](#).

Handling Conversion Errors

We suggest checking, which conversion errors are tolerable in your process and which must be considered critical. Set the property [ConversionErrorMask](#) to include critical errors only.

Conversion errors can often be resolved by optimizing the 3-Heights™ PDF to PDF/A Converter API's options and or installation:

1. Conversion errors can be minimized by converting to PDF/A-2 instead of PDF/A-1. PDF/A-2 allows some features of newer versions of the PDF Reference, e.g. transparency, optional content (layers), or embedded files.
2. If fonts were substituted, the missing fonts should be installed (see [Fonts](#)).
3. For documents with non convertible XMP Metadata it is recommended to update the PDF creating software to generate valid XMP metadata.
4. How signed documents can be converted to PDF/A is explained in the following [blog post by Dr. Hans Bärffuss](#).

In case of a conversion error, the output file is best presented to a user to decide whether or not the conversion result is acceptable. The property [ConversionErrors](#) is helpful to display a meaningful message, e.g. "Embedded files have been removed during PDF/A conversion." Also, all conversion events are written to the log file, indicating the cause of the conversion error.

For fully automated processes or documents which the user cannot accept, a fallback conversion can be added. For different conversion errors, different fallback conversions might be required:

1. If embedded files were removed, they can be extracted using the product 3-Heights™ PDF Extract and then converted to PDF/A by the 3-Heights™ PDF Document Converter.
2. Other conversion errors can be dealt with by creating an image based PDF using the 3-Heights™ PDF to Image Converter, which renders all pages and replaces their content with the resulting image. Optionally, some data such as bookmarks, links, or document metadata can be preserved.

5.1.3 Post Analysis

The post analysis step checks, whether or not the output file conforms to the requested standard. A post analysis error ([ErrorCode](#) is [PDF_E_POSTANALYSIS](#) see [TPDFErrorCode](#)) indicates that the output file is not PDF/A.

In case of a post analysis error, the conversion can be repeated with the [ReportDetails](#) property set. The log file then indicates why the post analysis failed. Often the issue can be resolved, e.g. by installing missing fonts (see [Fonts](#)) or with the [AllowDowngrade](#) property set.

If the file cannot be converted, a meaningful fallback could be the conversion to an image based PDF as described above in chapter [Handling Conversion Errors](#).

5.2 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 please visit: <http://www.pdf-tools.com/pdf20/en/resources/pdf-iso-standards/>.

5.2.1 PDF/A-1

The PDF/A-1 format is described in the international standard ISO-19005-1. It is based on the PDF 1.4 reference and has some additional requirements. It is beneficial to have a general understanding of PDF/A. Here is a brief overview of how to create a PDF/A document from a non-PDF/A document.

1. A PDF/A has requirements about meta data and the structure of the file. The PDF to PDF/A Converter takes care of this and the user does not have to apply any settings. However he can provide the XMP meta data himself if desired.
2. In PDF/A, colors (including grayscale and black/white) must not be represented in a device color space (DeviceRGB, DeviceCMYK, DeviceGray). Suitable default color space profiles to substitute the device color spaces, one for RGB, CMYK and grayscale respectively can be provided by the user. In addition, or alternatively, one color space profile can be embedded as output intent. In this latter case, device colors are automatically managed by the output intent if the color can be represented in the space given by the color space profile in the output intent.
If the converter encounters unmanaged colors, e.g. because no color space profile was set, then a calibrated color space is generated automatically, one RGB and one grayscale, for RGB and grayscale colors respectively. If unmanaged CMYK colors are encountered, a default CMYK output intent is embedded.
3. Fonts used in visible text must be embedded. This is automatically done by the Converter.
4. For PDF/A-1a: The original document structure information will be retained when converting the file to PDF/A. However, new tags will not be added and the structure will not be changed. To create a PDF/A-1a conforming file, the original file must have been created with the required structure and tagging. Otherwise, a PDF/A-1b file will be produced.

5.2.2 What is the difference between PDF/A-1b and PDF/A-1a?

PDF/A-1a has additional specifications on top of PDF/A-1b. These are:

1. The encoding of fonts must meet additional requirements, e.g. include a ToUnicode mapping (ISO 19005-1, chapter 6.3.8)
2. The document must contain a logical structure (ISO 19005-1, chapter 6.8)

The idea of the PDF/A-1a requirements is mainly to provide support for disabled people, i.e. by providing the required information needed for applications that support the read out loud feature.

The logical structure of the document is a description of the content of the pages. This description has to be provided by the creator of the document. It consists of a fine granular hierarchical tagging that distinguishes between the actual content and artifacts (such as page numbers, footers, layout artifacts, etc.). The tagging provides a meaningful description. Examples are "This is a Header", "This color image shows a small sailing boat at sunset", etc. One can easily understand this information cannot be generated automatically, it needs to be provided. This is one of the reasons why not every PDF document can be converted to PDF/A-1a.

5.2.3 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)
- 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.

5.2.4 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.

5.3 Color Spaces

5.3.1 Colors in PDF

The PDF format supports a range of color spaces:

Device Color Spaces (DeviceGray, DeviceRGB, and DeviceCMYK) These are also referred to as uncalibrated color spaces, because they cannot be used to specify color values such that colors are reproducible in a predictable way on multiple output devices.

CIE-based Color Spaces (CalGray, CalRGB, Lab, ICCBased) These are also referred to as device-independent color spaces, because they are inherently capable of specifying colors which can be reliably reproduced on multiple output devices.

Special Color Spaces (Separation and DeviceN) These require an alternate color space from one of the previous two groups to allow the PDF consumer to simulate the color on devices which do not support the special color space.

Colors can occur in the following objects of a PDF/A document:

- Raster images (also inline images)
- Text and Vector objects such as lines and curves
- Annotations
- Shading patterns
- Transparency blending (PDF/A-2 and later)

ICC Color Profiles

An ICC (International Color Consortium) profile is a file format which can be used to describe the color characteristics of a particular device. For example for the correct color reproduction when an image from a scanner or camera is displayed on a device, such as a monitor or printer. Color profiles are usually provided with the operating system (OS), on a Windows System, they can be found at the following location:

`%SystemRoot%\system32\spool\drivers\color`

Alternatively, additional profiles can be found here:

- <http://www.pdf-tools.com/public/downloads/resources/colorprofiles.zip>
- <http://www.color.org/srgbprofiles.html>
- https://www.adobe.com/support/downloads/iccprofiles/iccprofiles_win.html

Please note that most color profiles are copyrighted, therefore you should read the license agreements on the above links before using the color profiles. The PDF to PDF/A Converter will try to locate color profiles automatically in the %SystemRoot%\system32\spool\drivers\color folder as needed. On Linux or macOS, you can store the color profiles contained in the `colorprofiles.zip` download in a folder of your choice, and set the environment variable `PDF_ICC_PATH` to point to that folder.

PDF/A Requirements

In PDF/A the usage of uncalibrated color spaces (DeviceGray, DeviceRGB, and DeviceCMYK) is prohibited because colors that are specified in this way cannot be reproduced reliably on multiple output devices. Therefore, when converting to PDF/A, all device color spaces should be replaced by CIE-based color spaces. There is one exception to this rule: An uncalibrated color is tolerated if the output intent holds an ICC color profile with which this color can be represented. (E.g. a grayscale color can be represented in an RGB color profile, but a CMYK color cannot.)

The 3-Heights™ PDF to PDF/A Converter API uses the following strategy:

- For each device color space (DeviceGray, DeviceRGB, and DeviceCMYK) an ICC color profile can be specified to be used as substitute for the respective device color space.
- If an output intent is present, it is copied.
- Optionally an ICC color profile can be set to be used in the output intent.
- During conversion, if a device color space is encountered then the following is done:
 - If an output intent was set that is capable of managing this color, no action is needed.
 - Otherwise, if an ICC color profile is set to substitute this device color space then this color profile is used.
 - Otherwise, for DeviceRGB and DeviceGray color spaces: A calibrated color space (CalRGB⁹ and CalGray respectively) is generated and used as a substitute.
 - Otherwise, for DeviceCMYK color spaces:
 - If the output intent is not set, then a default CMYK ICC color profile is used for the output intent.
 - If the output intent holds a non-CMYK ICC color profile, then a default CMYK ICC color profile is generated and used as a substitute for DeviceCMYK.

The above strategy is motivated by the fact that CalRGB and CalGray color spaces occupy very little memory in comparison to ICC color profiles. Also note that the primary purpose of the output intent in a PDF document is to describe the characteristics of the device on which a document is intended to be rendered. Traditionally, the target device is a printer, which motivates CMYK output intents. The default CMYK color profile `USWebCoatedSWOP.icc` is provided in the sub-directory `bin\icc`.

5.4 Fonts

The PDF/A standard requires all fonts to be embedded in the PDF file. This ensures that the future rendering of the textual content of a conforming file matches, on a glyph by glyph basis, the appearance of the file as originally created.

Hence, if non-embedded fonts in a PDF are used, the font must be embedded. For this, a matching font has to be found in the [Font Directories](#). The method [AddFontDirectory](#) should be used to define additional directories. The default font directories are listed in the chapter [Font Directories](#).

It is important that the [Font Directories](#) contain all fonts that are used for the input files.

Fonts should be added to one of the [Font Directories](#), if the post analysis returns validation errors like the following:

```
"output.pdf", 9, 20, 0x00418704, "The font ShinGo must be embedded.", 1
```

⁹ The generated CalRGB color space is an approximation to the ICC color profile `sRGB Color Space Profile.icm`.

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

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

5.4.1 Font Cache

A cache of all fonts in all [Font Directories](#) is created. If fonts are added or removed from the font directories, the cache is updated automatically.

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

The font cache is created in the subdirectory `<CacheDirectory>/Installed Fonts` of the [Cache Directory](#).

5.4.2 Microsoft Core Fonts on Linux or macOS

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

Alternatively you can download the fonts from here:

<http://corefonts.sourceforge.net/>

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

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

5.4.3 Font Configuration File `fonts.ini`

The font configuration file is optional. It can be used to control the embedding of fonts.

The file `fonts.ini` must reside at the following location, which is platform dependent:

Windows: In a directory named `Fonts`, which must be a direct sub-directory of where `Pdf2PdfAPI.dll` resides.

Unix: The `fonts.ini` file is searched in the following locations

1. If the environment variable `PDFFONTDIR` is defined: `$PDFFONTDIR/fonts.ini`
2. `~/pdf-tools/fonts/fonts.ini`
3. `/etc/opt/pdf-tools/fonts/fonts.ini`

`fonts.ini` uses the INI file format and has two sections. The section `[fonts]` is ignored by the 3-Heights™ PDF to PDF/A Converter API, so you may remove it. In the section `[replace]` font replacement rules of the form `key=value` can be defined. The key specifies the font that is to be replaced. The key should match the name of the font mentioned in the pre-analysis of the 3-Heights™ PDF to PDF/A Converter API, e.g. **"ShingGo"** for:

```
"file.pdf", 9, 20, 0x00418704, "The font ShinGo must be embedded.", 1
```

The value should match the true type name of an installed font. Do not replace any standard fonts (Helvetica, Arial, Times, TimesNewRoman, Courier, CourierNew, Symbol, and ZapfDingbats).

Please note that this feature should be used with care. Replacing a font with another might change the visual appearance of the file because of different glyph shapes, metrics or glyphs that are not available in the replacement font. Embedding another font might also have legal implications.

Example: Replace MS-Mincyo with MS-Mincho

```
[replace]  
MS-Mincyo=MS-Mincho
```

This rule defines, that in order to embed a font program for font MS-Mincyo the font MS-Mincho should be used. This rule is useful, because both names are possible transliterations of the same Japanese font. However, the official transliteration used by the actual font is MS-Mincho.

5.5 Cryptographic Provider

In order to use the 3-Heights™ PDF to PDF/A Converter API's cryptographic functions such as creating digital signatures, a cryptographic provider is required. The cryptographic provider manages certificates, their private keys and implements cryptographic algorithms.

The 3-Heights™ PDF to PDF/A Converter API can use various different cryptographic providers. The following list shows, for which type of signing certificate which provider can be used.

USB Token or Smart Card These devices typically offer a PKCS#11 interface, which is the recommended way to use the certificate → [PKCS#11 Provider](#).

On Windows, the certificate is usually also available in the [Windows Cryptographic Provider](#).

Note that in any case, signing documents is only possible in an interactive user session.

Hardware Security Module (HSM) HSMs always offer very good PKCS#11 support → [PKCS#11 Provider](#)

For more information and installation instructions consult the separate document [TechNotePKCS11.pdf](#).

Soft Certificate Soft certificates are typically PKCS#12 files that have the extension `.pfx` or `.p12` and contain the signing certificate as well as the private key and trust chain (issuer certificates). Soft certificate files cannot be used directly. Instead, they must be imported into the certificate store of a cryptographic provider.

- *All Platforms:* The recommended way of using soft certificates is to import them into a store that offers a PKCS#11 interface and use the [PKCS#11 Provider](#). For example:
 - A HSM
 - openCryptoki on Linux

For more information and installation instructions of the above stores consult the separate document [TechNotePKCS11.pdf](#).

- *Windows:* If no PKCS#11 provider is available, soft certificates can be imported into Windows certificate store, which can then be used as cryptographic provider → [Windows Cryptographic Provider](#)

Signature Service Signature services are a convenient alternative to storing certificates and key material locally. The 3-Heights™ PDF to PDF/A Converter API can use various different services whose configuration is explained in the following sections of this documentation:

- [SwissSign Digital Signing Service](#)
- [Swisscom All-in Signing Service](#)
- [GlobalSign Digital Signing Service](#)
- [QuoVadis sealsign](#)

5.5.1 PKCS#11 Provider

PKCS#11 is a standard interface offered by most cryptographic devices such as HSMs, USB Tokens or sometimes even soft stores (e.g. openCryptoki).

More information on and installation instructions of the PKCS#11 provider of various cryptographic devices can be found in the separate document [TechNotePKCS11.pdf](#).

Configuration

Provider Property [Provider](#) or argument of [BeginSession](#)

The provider configuration string has the following syntax:

"<PathToDll>;<SlotId>;<Pin>"

<PathToDll> is the path to driver library filename, which is provided by the manufacturer of the HSM, UBS token or smart card. Examples:

- The CardOS API from Atos (Siemens) uses `siicap11.dll`
- The IBM 4758 cryptographic coprocessor uses `cryptoki.dll`
- Devices from Aladdin Ltd. use `etpkcs11.dll`
- The SuisselD USB Tokens use `cvP11.dll`

Please note that the sale of SuisselD will be discontinued as of 31. December 2019. On 15. December 2021 the SuisselD certificates will be revoked for regulatory reasons.

<SlotId> is optional, if it is not defined, it is searched for the first slot that contains a running token.

<Pin> is optional, if it is not defined, the submission for the pin is activated via the pad of the token.

If this is not supported by the token, the following error message is raised when signing: "Private key not available."

Example:

```
Provider = "C:\Windows\system32\siicap11.dll;4;123456"
```

Note: Some PKCS#11 drivers require the [Terminate](#) method to be called. Otherwise your application might crash upon termination.

The chapter [Guidelines for Mass Signing](#) contains important information to optimize performance when signing multiple documents.

Interoperability Support

The following cryptographic token interface (PKCS#11) products have been successfully tested:

- SafeNet Protect Server
- SafeNet Luna
- SafeNet Authentication Client
- IBM OpenCrypTokI
- CryptoVision
- Siemens CardOS
- Utimaco SafeGuard CryptoServer

Selecting a Certificate for Signing

The 3-Heights™ PDF to PDF/A Converter API offers different ways to select a certificate. The product tries the first of the following selection strategies, for which the required values have been specified by the user.

1. Certificate fingerprint

Property [SignerFingerprint](#)

- SHA1 fingerprint of the certificate. The fingerprint is 20 bytes long and can be specified in hexadecimal string representation, e.g. "b5 e4 5c 98 5a 7e 05 ff f4 c6 a3 45 13 48 0b c6 9d e4 5d f5". In Windows certificate store this is called "Thumbprint", if "Thumbprint algorithm" is "sha1".

2. Certificate Issuer and SerialNumber

Properties [Issuer](#) and [SerialNumber](#)

- Certificate Issuer (e.g. "QV Schweiz CA"), in Windows certificate store this is called "Issued By".
- Serial number of the certificate (hexadecimal string representation, e.g. "4c 05 58 fb"). This is a unique number assigned to the certificate by its issuer. In Windows certificate store this is the field called "Serial number" in the certificate's "Details" tab.

3. Certificate Name and optionally Issuer

Properties [Name](#) and [Issuer](#)

- Common Name of the certificate (e.g. "PDF Tools AG"), in Windows certificate store this is called "Issued To".
- Optional: Certificate Issuer (e.g. "QV Schweiz CA"), in Windows certificate store this is called "Issued By".

Using PKCS#11 stores with missing issuer certificates

Some PKCS#11 devices contain the signing certificate only. However, in order to embed revocation information it is important, that the issuer certificates, i.e. the whole trust chain, is available as well.

On Windows, missing issuer certificates can be loaded from the Windows certificate store. So the missing certificates can be installed as follows:

1. Get the certificates of the trust chain. You can download them from the website of your certificate provider or do the following:
 - a. Sign a document and open the output in Adobe Acrobat
 - b. Go to "Signature Properties" and then view the signer's certificate
 - c. Select a certificate of the trust chain
 - d. Export the certificate as "Certificate File" (extension .cer)
 - e. Do this for all certificates of the trust chain
2. Open the exported files by double clicking on them in the Windows Explorer
3. Click button "Install Certificate..."
4. Select "automatically select the certificate store based on the type of certificate" and finish import

Cryptographic Suites

Message Digest Algorithm

The default hash algorithm to create the message digest is **SHA-256**. Other algorithms can be chosen by setting the provider session property [MessageDigestAlgorithm](#), for which supported values are:

SHA-1 This algorithm is considered broken and therefore strongly discouraged by the cryptographic community.

SHA-256 (default)

SHA-384

SHA-512

RIPEMD-160

Signing Algorithm

The signing algorithm can be configured by setting the provider session property [SigAlgo](#). Supported values are:

RSA_RSA (default) This is the RSA PKCS#1v1.5 algorithm which is widely supported by cryptographic providers.

RSA_SSA_PSS This algorithm is sometimes also called RSA-PSS.

Signing will fail if the algorithm is not supported by the cryptographic hardware. The device must support either the signing algorithm CKM_RSA_PKCS_PSS (i.e. RSA_SSA_PSS) or CKM_RSA_X_509 (i.e. raw RSA).

Note: Setting the signing algorithm only has an effect on signatures created by the cryptographic provider itself. All signed data acquired from external sources might use other signing algorithms, specifically the issuer signatures of the trust chain, the time-stamp's signature, or those used for the revocation information (CRL, OCSP). It is recommended to verify, that the algorithms of all signatures provide a similar level of security.

5.5.2 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).

Microsoft CryptoAPI Provides functionality for using cryptographic algorithms and for accessing certificates stored in the Windows certificate store and other devices, such as USB tokens, with Windows integration.

Microsoft CryptoAPI does not support some new cryptographic algorithms, such as SHA-256.

Cryptography API: Next Generation (CNG) CNG is an update to CryptoAPI. It extends the variety of available cryptographic algorithms, e.g. by the SHA-256 hashing algorithms. If possible the 3-Heights™ PDF to PDF/A Converter API performs cryptographic calculations with CNG instead of CryptoAPI.

CNG is available only if:

- The operating system is at least Windows Vista or Windows Server 2008.
- The provider of the signing certificate's private key, e.g. the USB Token or SmartCard, supports CNG.

If CNG is not available, the CryptoAPI's cryptographic algorithms are used. In any case, CryptoAPI is used for the certificate accessing functionalities.

Default Message Digest Algorithm: Since version 4.6.12.0 of the 3-Heights™ PDF to PDF/A Converter API, the default message digest algorithm is SHA-256. As a result, signing will fail if CNG is not available (error message "Private key not available."). To use SHA-1, the provider session property [MessageDigestAlgorithm](#) can be used. Note that the use of SHA-1 is strongly discouraged by the cryptographic community.

Configuration

Provider Property [Provider](#) or argument of [BeginSession](#)

The provider configuration string has the following syntax:

```
"[<ProviderType>:]<Provider>[;<PIN>]"
```

The <ProviderType> and <PIN> are optional. The corresponding drivers must be installed on Windows. If CNG is available, <ProviderType> and <Provider> are obsolete and can be omitted.

Optionally, when using an advanced certificate, the pin code (password) can be passed as an additional, semi-column separated parameter <PIN>. This does not work with qualified certificates, because they always require the pin code to be entered manually and every time.

If <Provider> is omitted, the default provider is used. The default provider is suitable for all systems where CNG is available.

Examples: Use the default provider with no pin.

```
Provider = ""
```

Examples: "123456" being the pin code.

```
Provider = ";123456"
```

```
Provider = "Microsoft Base Cryptographic Provider v1.0;123456"
```

```
Provider = "PROV_RSA_AES:Microsoft Enhanced RSA and AES Cryptographic" _  
+ "Provider;123456"
```

Certificate Store [Property Store](#)

The value for the certificate store depends on the OS. Supported values are: "CA", "MY" and "ROOT". For signature creation the default store "MY" is usually the right choice.

Store Location [Property StoreLocation](#)

Either of the following store locations

- "Local Machine"
- "Current User" (default)

Usually personal certificates are stored in the "Current User" location and company-wide certificates are stored under "Local Machine".

The "Current User"'s store is only available, if the user profile has been loaded. This may not be the case in certain environments such as within an IIS web application or COM+ applications. Use the store of the Local Machine, if the user profile cannot be loaded. For other services it is sufficient to log it on as the user. Note that some cryptographic hardware (such as smart cards or USB Tokens) require an interactive environment. As a result, the private key might not be available in the service session, unless the 3-Heights™ PDF to PDF/A Converter API is run interactively.

Certificates in the store "Local Machine" are available to all users. However, in order to sign a document, you need access to the signing certificate's private key. The private key is protected by Windows ACLs and typically readable for Administrators only. Use the Microsoft Management Console (`mmc.exe`) in order to grant access to the private key for other users as follows: Add the Certificates Snap-in for the certificates on Local Machine. Right-click on the signing certificate, click on "All Tasks" and then "Manage Private Keys..." where you can set the permissions.

Selecting a Certificate for Signing

Within the certificate store selected by [Store Location](#) and [Certificate Store](#) the selection of the signing certificate works the same as with the PKCS#11 provider, which is described here: [Selecting a Certificate for Signing](#)

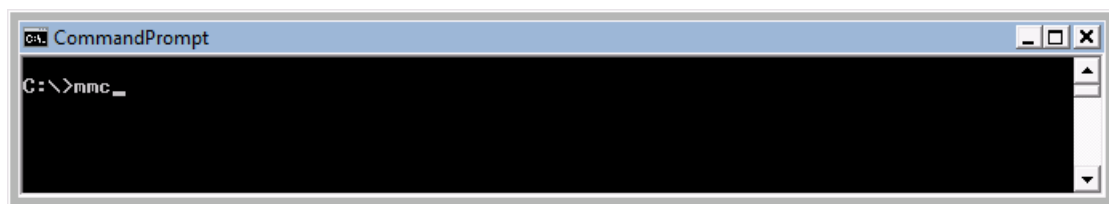
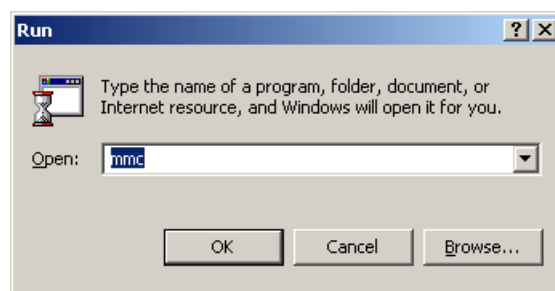
Certificates

In order to sign a PDF document, a valid, existing certificate name must be provided and its private key must be available.

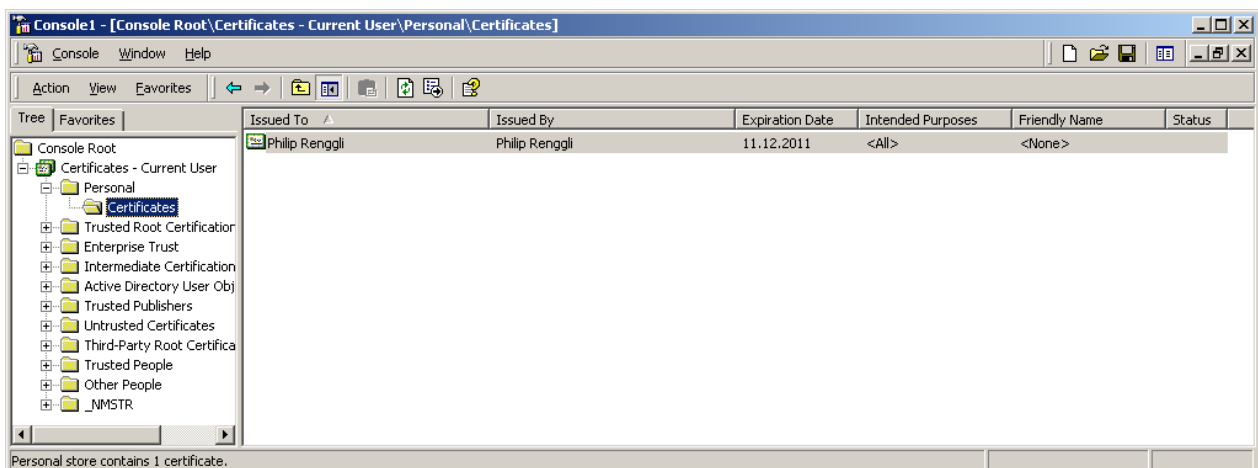
There are various ways to create or obtain a certificate. How this is done is not described in this document. This document describes the requirements for, and how to use the certificate.

On the Windows operating system certificates can be listed by the Microsoft Management Console (MMC), which is provided by Windows. In order to see the certificates available on the system, do the following steps:

1. To launch the MMC, go to Start → Run... → type "mmc", or start a Command Prompt and type "mmc".



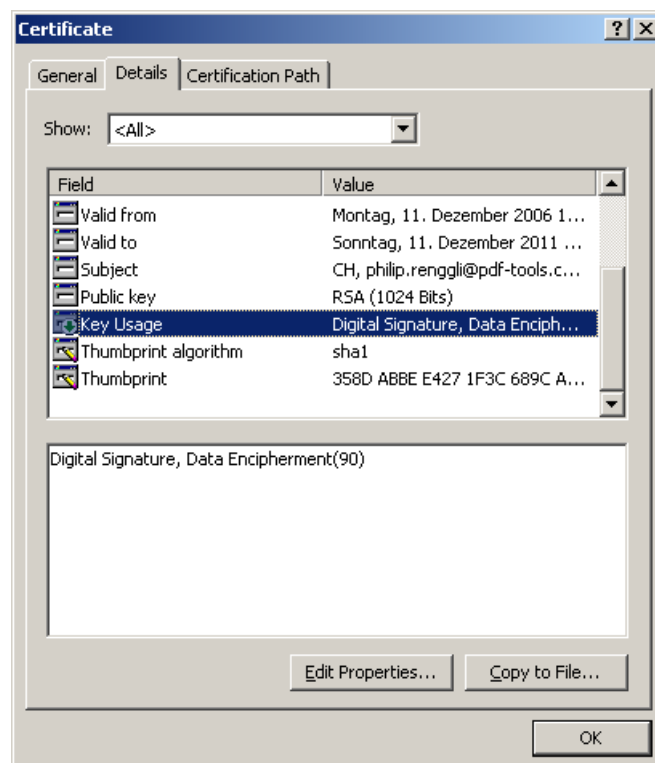
2. Under "File" → "Add/Remove Snap-in"
3. Choose "Certificates" and click the "Add" button
4. In the next window choose to manage certificates for "My user account"
5. Click "Finish"
6. The certificate must be listed under the root "Certificates - Current User", for example as shown in the screenshot below:



7. Double-click the certificate to open. The certificate name corresponds to the value “Issued to:”.

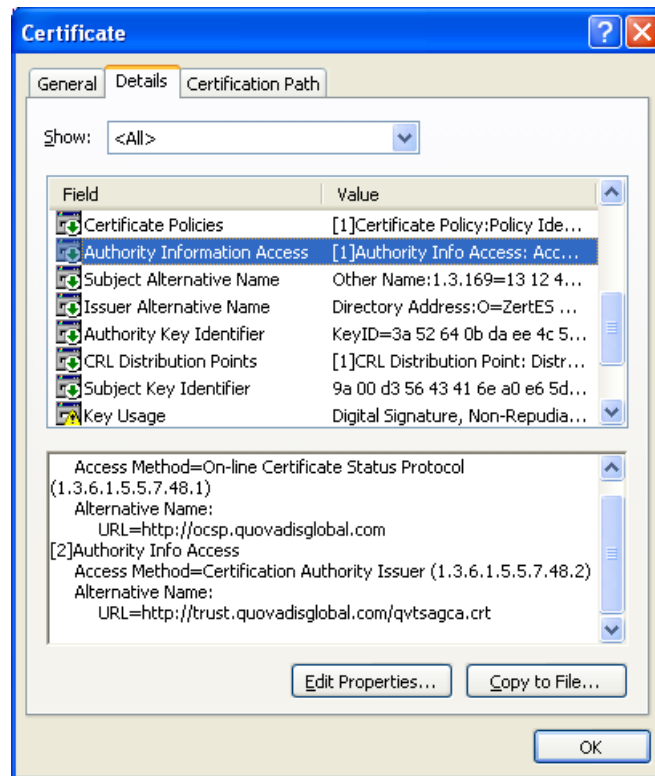


8. In the tab Detail of the certificate, there is a field named “Key Usage”. This field must contain the value “Digital Signature”. Additional values are optional, see also screenshot. You must have the private key that corresponds to this certificate.



Qualified Certificates

A qualified certificate can be obtained from a certificate authority (CA). Besides the requirements listed in the previous chapter it has the additional requirement to contain the key “Authority Information Access” which contains the information about the OCSP server.



Cryptographic Suites

The message digest algorithm as well as the signing algorithm can be chosen as described for the PKCS#11 provider in [Cryptographic Suites](#).

The `MessageDigestAlgorithm` can only be set to a value other than **SHA-1** if the private key's provider supports CNG.

The `SigAlgo` can only be set to **RSA_SSA_PSS** if the private key's provider supports CNG.

5.5.3 SwissSign Digital Signing Service

Provider Property [Provider](#) or argument of [BeginSession](#)

The provider configuration string contains the URL to the service endpoint.

Provider Configuration The provider can be configured using provider session properties.

There are two types of properties:

- “String” Properties:
String properties are set using method [SetSessionProperty](#).
- “File” Properties:
File properties are set using method [SetSessionProperty](#) with a file name parameter. Alternatively the file can be passed in-memory as byte array using the method [SetSessionProperty](#).

Name	Type	Required	Value
Identity	String	required	The identity of your signing certificate. Example: My Company:Signing Cert 1
DSSProfile	String	required	Must be set to http://dss.swissign.net/dss/profile/pades/1.0
SSLClientCertificate	File	required	SSL client certificate in PKCS#12 Format (.p12, .pfx). File must contain the certificate itself, all certificates of the trust chain and the private key.
SSLClientCertificatePassword	String	optional	Password to decrypt the private key of the SSL client certificate.
SSLServerCertificate	File	recommended	Certificate of the server or its issuer (CA) certificate (.crt). The certificate may be in either PEM (ASCII text) or DER (binary) form. Note: If this property is not set, the server certificate's trustworthiness cannot be determined. As a result, the connection is not guaranteed to be secure.
RequestID	String	recommended	Any string that can be used to track the request. Example: An UUID like AE57F021-C0EB-4AE0-8E5E-67FB93E5BC7F

Signature Configuration The signature can be customized using standard properties of the 3-Heights™ PDF to PDF/A Converter API.

Description	Required	Value	Setting
Common Name	required	The name of the signer must be set ¹⁰ .	Property Name .
Time-stamp	optional	Use the value urn:ietf:rfc:3161 to embed a time-stamp.	Property TimeStampURL
Signature Format	optional	To set the signature format	Property SubFilter . Must be adbe.pkcs7.detached
Revocation Info	recommended	To embed OCSP responses or CRL.	Property EmbedRevocationInfo

Visual Appearance	optional	See separate chapter How to Create a Visual Appearance of a Signature .
--------------------------	----------	---

Proxy Configuration If a proxy is used for the connection to the service, see chapter [How to Use a Proxy](#) for more information.

5.5.4 QuoVadis sealsign

Provider Property [Provider](#) or argument of [BeginSession](#)

The provider configuration string contains the URL to the QuoVadis sealsign service.

- Demo service:
<https://services.sealsignportal.com/sealsign/ws/BrokerClient>
- Productive service:
<https://qvchsvsws.quovadisglobal.com/sealsign/ws/BrokerClient>

Provider Configuration The provider can be configured using provider session properties that can be set using the method [SetSessionProperty](#).

Name	Type	Required	Value
Identity	String	required	The account ID is the unique name of the account specified on the server. Example: Rigora
Profile	String	required	The profile identifies the signature specifications by a unique name. Example: Default
secret	String	required	The secret is the password which secures the access to the account. Example: NeE=EKEd33FeCk70
clientId	String	required	A client ID can be used to help separating access and creating better statistics. If specified in the account configuration it is necessary to provide this value. Example: 3949-4929-3179-2818
pin	String	required	The PIN code is required to activate the signing key. Example: 123456

¹⁰ This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

MessageDigestAlgorithm	String	optional	The message digest algorithm to use. Default: SHA-256 Alternatives: SHA-1 , SHA-384 , SHA-512 , RIPEMD-160 , RIPEMD-256
-------------------------------	--------	----------	---

Signature Configuration The signature can be customized using standard properties.

Description	Required	Value	Setting
Common Name	required	The name of the signer must be set ¹¹ .	Property Name .
Time-stamp	-	Not available.	
Revocation Info	recommended	To embed OCSP responses or CRL.	Property EmbedRevocationInfo
Visual Appearance	optional	See separate chapter How to Create a Visual Appearance of a Signature .	

Proxy Configuration If a proxy is used for the connection to the service, see chapter [How to Use a Proxy](#) for more information.

5.5.5 Swisscom All-in Signing Service

General Properties

To use the signature service, the following general properties have to be set:

Description	Required	Value	Setting
Common Name	required	Name of the signer ¹² .	Property Name
Provider	required	The service endpoint URL of the REST service. Example: https://ais.swisscom.com/AIS-Server/rs/v1.0/sign	Property Provider
Time-stamp	optional	Use the value urn:ietf:rfc:3161 to embed a time-stamp.	Property TimeStampURL

¹¹ This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

Signature Format	optional	To set the signature format	Property SubFilter . Supported values are adbe.pkcs7.detached , ETSI.CAdES.detached .
Revocation Info	optional	To embed OCSP responses	Property EmbedRevocationInfo . Supported with adbe.pkcs7.detached only.

If a proxy is used for the connection to the service, see chapter [How to Use a Proxy](#) for more information.

Provider Session Properties

In addition to the general properties, a few provider specific session properties have to be set.

There are two types of properties:

- “String” Properties:
String properties are set using method [SetSessionProperty](#).
- “File” Properties:
File properties are set using method [SetSessionProperty](#) with a file name parameter. Alternatively the file can be passed in-memory as byte array using the method [SetSessionProperty](#).

Name	Type	Required	Value
DSSProfile	String	required	Must be set to http://ais.swisscom.ch/1.0
SSLClientCertificate	File	required	SSL client certificate in PKCS#12 Format (.p12, .pfx). File must contain the certificate itself, all certificates of the trust chain and the private key.
SSLClientCertificatePassword	String	optional	Password to decrypt the private key of the SSL client certificate.
SSLServerCertificate	File	recommended	Certificate of the server or its issuer (CA) certificate (.crt). The certificate may be in either PEM (ASCII text) or DER (binary) form. Note: If this property is not set, the server certificate's trustworthiness cannot be determined. As a result, the connection is not guaranteed to be secure.
Identity	String	required	The Claimed Identity string as provided by Swisscom: <customer name>:<key identity>

¹² This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

RequestID	String	recommended	Any string that can be used to track the request. Example: An UUID like AE57F021-C0EB-4AE0-8E5E-67FB93E5BC7F
------------------	--------	-------------	---

On-Demand Certificates

To request an on-demand certificate, the following additional property has to be set:

Name	Type	Required	Value
SwisscomAllInOnDemandDN	String	required	The requested distinguished name. Example: cn=Hans Muster,o=ACME,c=CH

Step-Up Authorization using Mobile-ID

To use the step-up authorization, the following additional properties have to be set:

Name	Type	Required	Value
SwisscomAllInMSISDN	String	required	Mobile phone number. Example: +41798765432
SwisscomAllInMessage	String	required	The message to be displayed on the mobile phone. Example: Pipapo halolu.
SwisscomAllInLanguage	String	required	The language of the message. Example: DE

Those properties have to comply with the Swisscom Mobile-ID specification.

5.5.6 GlobalSign Digital Signing Service

Provider Property [Provider](#) or argument of [BeginSession](#)

The provider configuration string contains the URL to the service endpoint.

<https://emea.api.dss.globalsign.com:8443/v2>

Provider Configuration The provider can be configured using provider session properties.

There are two types of properties:

- "String" Properties:
String properties are set using method [SetSessionProperty](#).
- "File" Properties:
File properties are set using method [SetSessionProperty](#) with a file name parameter. Alternatively the file can be passed in-memory as byte array using the method [SetSessionProperty](#).

Name	Type	Required	Value
api_key	String	required	Your account credentials' key parameter for the login request.
api_secret	String	required	Your account credentials' secret parameter for the login request.
Identity	String	required	Parameter to create the signing certificate. Example for an account with a static identity: <code>{}</code> Example for an account with a dynamic identity: <code>{ "subject_dn": { "common_name": "John Doe" } }</code>
SSLClientCertificate	File	required	SSL client certificate in PKCS#12 Format (.p12, .pfx). File must contain the certificate itself, all certificates of the trust chain and the private key.
SSLClientCertificatePassword	String	optional	Password to decrypt the private key of the SSL client certificate.
SSLServerCertificate	File	recommended	Certificate of the server or its issuer (CA) certificate (.cer). The certificate may be in either PEM (ASCII text) or DER (binary) form. Note: If this property is not set, the server certificate's trustworthiness cannot be determined. As a result, the connection is not guaranteed to be secure.

Signature Configuration The signature can be customized using standard properties of the 3-Heights™ PDF to PDF/A Converter API.

Description	Required	Value	Setting
Common Name	required	The name of the signer must be set ¹³ .	Property Name .
Time-stamp	recommended	Use the value <code>urn:ietf:rfc:3161</code> to embed a time-stamp.	Property TimeStampURL
Signature Format	optional	To set the signature format	Property SubFilter . Supported values are <code>adbe.pkcs7.detached</code> , <code>ETSI.CAdES.detached</code> .

Revocation Info	recommended	To embed OCSP responses or CRL.	Property EmbedRevocationInfo
Visual Appearance	optional	See separate chapter How to Create a Visual Appearance of a Signature .	

Proxy Configuration If a proxy is used for the connection to the service, see chapter [How to Use a Proxy](#) for more information.

How to create the SSL client certificate

When creating a new account, GlobalSign will issue an SSL client certificate `clientcert.crt`. The following command creates a PKCS#12 file `certificate.p12` that can be used for the [SSLClientCertificate](#):

```
openssl pkcs12 -export -out certificate.p12 -inkey privateKey.key -in clientcert.crt
```

How to get the SSL server certificate

The SSL server certificate can either be found in the technical documentation of the “Digital Signing Service” or downloaded from the server itself:

1. Get the server’s SSL certificate:

```
openssl s_client -showcerts -connect emea.api.dss.globalsign.com:8443 ^
-cert clientcert.crt -key privateKey.key
```

2. The certificate is the text starting with “-----BEGIN CERTIFICATE-----” and ending with “-----END CERTIFICATE-----”. Use the text to create a text file and save it as `server.crt`.
3. Use `server.crt` or one of its CA certificates for the [SSLServerCertificate](#).

Advice on using the service

Whenever a new session is created using [BeginSession](#) a login is performed. In this session signatures can be created using different identities, i.e. signing certificates, which are created as they are needed. Both signing sessions and signing certificates expire after 10 minutes.

Note that there are rate limits for both creating new identities and for signing operations. So, if multiple documents must be signed at once, it is advisable to re-use the same session (and hence its signing certificates) for signing.

Due to the short-lived nature of the signing certificates, it is important to embed revocation information immediately. For example by using [AddValidationInformation\(\)](#) of the 3-Heights™ PDF Security API or [EmbedRevocationInfo](#). Furthermore it is highly recommended to embed a time-stamp in order to prove that the signature was created during the certificate’s validity period.

5.6 How to Create Digital Signatures

This chapter describes the steps that are required to create different types of digital signatures. A good introductory example can be found in the chapter [How to Create Electronic Signatures](#).

¹³ This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

5.6.1 How to Create a PAdES Signature

The PAdES European Norm (ETSI EN 319 142) recommends to use one of the following four baseline signature levels.

PAdES-B-B A digital signature.

PAdES-B-T A digital signature with a time-stamp token.

PAdES-B-LT A digital signature with a time-stamp token and signature validation data. The signature is a long-term signature or "LTV enabled".

PAdES-B-LTA A digital signature with a time-stamp token and signature validation data protected by a document time-stamp.

The lifecycle of digital signatures in general and usage these signature levels in particular are described in more detail in chapter 8.11.6 "Digital signatures lifecycle" of ETSI TR 119 100.

Note: The Decision 2015/1506/EU of the eIDAS Regulation (Regulation (EU) N°910/2014) still refers to the previous legacy PAdES baseline signature standard ETSI TS 103 172. However, the signatures as created by the 3-Heights™ PDF to PDF/A Converter API are compatible.

The [Compatibility of PAdES Signature Levels](#) shows, to which other standards the signature levels described above and as created by the 3-Heights™ PDF to PDF/A Converter API conform.

Compatibility of PAdES Signature Levels

ETSI EN 319 142	ETSI TS 102 778	ETSI TS 103 172	ISO 14533-3
PAdES-B-B	PAdES-BES (Part 3)	PAdES B-Level	-
PAdES-B-T	PAdES-BES (Part 3)	PAdES T-Level	PAdES-T
PAdES-B-LT	PAdES-BES (Part 3)	PAdES LT-Level	PAdES-A
PAdES-B-LTA	PAdES-LTV (Part 4)	PAdES LTA-Level	PAdES-A

Requirements

For general requirements and preparation steps see chapter [How to Create Electronic Signatures](#).

Requirements

Level	Signing Certificate	Time-stamp	Product
PAdES-B-B	any	no	3-Heights™ PDF to PDF/A Converter API
PAdES-B-T	any	required	3-Heights™ PDF to PDF/A Converter API
PAdES-B-LT	advanced or qualified certificate	required	3-Heights™ PDF Security
PAdES-B-LTA	advanced or qualified certificate	required	3-Heights™ PDF Security

Make sure the trust store of your cryptographic provider contains all certificates of the trust chain, including the root certificate. Also include the trust chain of the time-stamp signature, if your TSA server does not include them in the time-stamp.

A proper error handling is crucial in order to ensure the creation of correctly signed documents. The output document was signed successfully, if and only if the method [Convert](#), [ConvertMem](#), [ConvertStream](#) returns true.

Note on linearization: Because signature levels PAdES-B-LT and PAdES-B-LTA must be created in a two-step process, the files cannot be linearized. When creating signature levels PAdES-B-B or PAdES-B-T that might later be augmented, linearization should not be used.

PAdES vs. CAdES: CAdES is an ETSI standard for the format of digital signatures. The format used in PAdES is based on CAdES, which is why the format is called **ETSI.CAdES.detached** (see [SubFilter](#)). Because PAdES defines additional requirements suitable for PDF signatures, mere CAdES conformance is not sufficient.

Create a PAdES-B-B Signature

Input Document Any PDF document.

Cryptographic Provider A cryptographic provider that supports the creation of PAdES signatures.

```
using (Pdf2Pdf doc = new Pdf2Pdf())
{
    if (!doc.BeginSession(@"myPKCS11.dll;0;pin"))
        throw new Exception("Error connecting to provider: " + doc.ErrorMessage);

    using (Signature sig = new Signature())
    {
        sig.Name = "My Signing Certificate";
        sig.SubFilter = "ETSI.CAdES.detached";
        sig.EmbedRevocationInfo = false;
        doc.AddSignature(sig);
    }

    if (!doc.Convert("input.pdf", "", "pades-b-b.pdf", "log.txt"))
        throw new Exception("Error saving pades-b-b.pdf: " + doc.ErrorMessage);
}
```

Create a PAdES-B-T Signature

Input Document Any PDF document.

Cryptographic Provider A cryptographic provider that supports the creation of PAdES signatures.

```
using (Pdf2Pdf doc = new Pdf2Pdf())
{
```

```

if (!doc.BeginSession(@"myPKCS11.dll;0;pin"))
    throw new Exception("Error connecting to provider: " + doc.ErrorMessage);

using (Signature sig = new Signature())
{
    sig.Name = "My Signing Certificate";
    sig.SubFilter = "ETSI.CAdES.detached";
    sig.EmbedRevocationInfo = false;
    sig.TimestampURL = "http://server.mydomain.com/tsa";
    doc.AddSignature(sig);
}

if (!doc.Convert("input.pdf", "", "pades-b-t.pdf", "log.txt"))
    throw new Exception("Error converting pades-b-t.pdf: " + doc.ErrorMessage);
}

```

5.6.2 How to Create a Visual Appearance of a Signature

Each signature may have a visual appearance on a page of the document. The visual appearance is optional and has no effect on the validity of the signature. Because of this and because a visual appearance may cover important content of the page, the 3-Heights™ PDF to PDF/A Converter API creates invisible signatures by default.

In order to create a visual appearance, a non-empty signature rectangle must be set. For example, by setting the property [Rect](#) to `[10, 10, 210, 60]` the following appearance is created:



Different properties of the visual appearance can be specified.

Page and Position See properties [PageNo](#) and [Rect](#).

Color See properties [FillColor](#) and [StrokeColor](#).

Line Width The line width of the background rectangle, see property [LineWidth](#).

Text Two text fragments can be set using two different fonts, font sizes, and colors see properties [Text1](#), [Text2](#), [FontName1](#), [FontName2](#), [FontSize1](#), and [FontSize2](#).

Background image See property [ImageFileName](#).

5.6.3 Guidelines for Mass Signing

This section provides some guidelines for mass signing using the 3-Heights™ PDF to PDF/A Converter API.

Keep the session to the security device open for multiple sign operations

Creating and ending the session to the security device is a complex operation. By re-using the session for multiple sign operations, performance can be improved:

1. Create a [Pdf2Pdf](#) object.
2. Open the session to the provider using [BeginSession](#).

3. Use the [Pdf2Pdf](#) object to sign multiple documents.
4. Close the session to the provider using [EndSession](#).
5. Dispose of the [Pdf2Pdf](#) object.

Signing concurrently using multiple threads

The 3-Heights™ PDF to PDF/A Converter API is thread-safe. Each [Pdf2Pdf](#) object should be used in one thread at the time only. It is recommended that each thread has a separate [Pdf2Pdf](#) object.

The performance improvement when signing concurrently using multiple threads depends mainly on the security device used. Typically the improvement is large for HSMs and small for USB Tokens.

Thread safety with a PKCS#11 provider

The PKCS#11 standard specifies, that “an application can specify that it will be accessing the library concurrently from multiple threads, and the library must [...] ensure proper thread-safe behavior.” However, some PKCS#11 provider (middleware) implementations are not thread-safe. For this reason, the 3-Heights™ PDF to PDF/A Converter API synchronizes all access to the same provider (middleware and slot id).

If your middleware is thread-safe, you can enable full parallel usage of the cryptographic device by setting the session property "[LOCKING_OK](#)" to the value "[True](#)" using the method [SetSessionProperty](#).

Example: Enable parallel access to the cryptographic device.

```
doc.SetSessionPropertyString("LOCKING_OK", "true");
```

5.6.4 Miscellaneous

Caching of CRLs, OCSP, and Time-stamp Responses

In order to improve the speed when mass signing, the 3-Heights™ PDF to PDF/A Converter API provides a caching algorithm to store CRL (Certificate Revocation List), OCSP (Online Certificate Status Protocol), TSP (Time-stamp Protocol) and data from signature services. This data is usually valid over period of time that is defined by the protocol, which is normally at least 24 hours. Caching improves the speed, because there are situations when the server does not need to be contacted for every digital signature.

The following caches are stored automatically by the 3-Heights™ PDF to PDF/A Converter API at the indicated locations within the [Cache Directory](#):

Certificates	<CacheDirectory>/Certificates/hash.cer
CRL	<CacheDirectory>/CLRs/server.der
OCSP responses	<CacheDirectory>/OCSP Responses/server-hash.der
Service data	<CacheDirectory>/Signature Sizes/hash.bin
Time-stamp responses ¹⁴	<CacheDirectory>/Time Stamps/server.der

¹⁴ The sizes of the time-stamp responses are cached only. Cached Time stamp responses cannot be embedded but used for the computation of the signature length only.

The caches can be cleared by deleting the files. Usage of the caches can be deactivated by setting the [NoCache](#) flag. The files are automatically updated if the current date and time exceeds the “next update” field in the OCSP or CRL response respectively or the cached data was downloaded more than 24 hours ago.

How to Use a Proxy

The 3-Heights™ PDF to PDF/A Converter API can use a proxy server for all communication to remote servers, e.g. to download CRL or for communication to a signature service. The proxy server can be configured using the provider session property [Proxy](#). The property's value must be a string with the following syntax:

```
http[s]://[<user>[:<password>]@<host>[:<port>]
```

Where:

- [http](#) / [https](#): Protocol for connection to proxy.
- [<user>: <password>](#) (optional): Credentials for connection to proxy (basic authorization).
- [<host>](#): Hostname of proxy.
- [<port>](#): Port for connection to proxy.

For SSL connections, e.g. to a signature service, the proxy must allow the HTTP CONNECT request to the signature service.

Example: Configuration of a proxy server that is called “myproxy” and accepts HTTP connections on port 8080.

```
conv.SetSessionPropertyString("Proxy", "http://myproxy:8080")
```

Configuration of Proxy Server and Firewall

For the application of a time-stamp or online verification of certificates, the signature software requires access to the server of the certificates' issuer (e.g. <http://ocsp.quovadisglobal.com> or <http://platinum-qualified-g2.ocsp.swissign.net/>) via HTTP. The URL for verification is stored in the certificate; the URL for time-stamp services is provided by the issuer. In case these functions are not configured, no access is required.

In organizations where a web proxy is used, it must be ensured that the required MIME types are supported. These are:

OCSP

- [application/ocsp-request](#)
- [application/ocsp-response](#)

Time-stamp

- [application/timestamp-query](#)
- [application/timestamp-reply](#)

Signature services

- Signature service specific MIME types.

Setting the Signature Build Properties

In the signature build properties dictionary the name of the application that created the signature can be set using the provider session properties [Prop_Build.App.Name](#) and [Prop_Build.App.REx](#). The default values are “3-Heights™ PDF to PDF/A Converter API” and its version.

5.7 How to Validate Digital Signatures

5.7.1 Validation of a Qualified Electronic Signature

There are basically three items that need to be validated:

1. Trust Chain
2. Revocation Information (optional)
3. Time-stamp (optional)

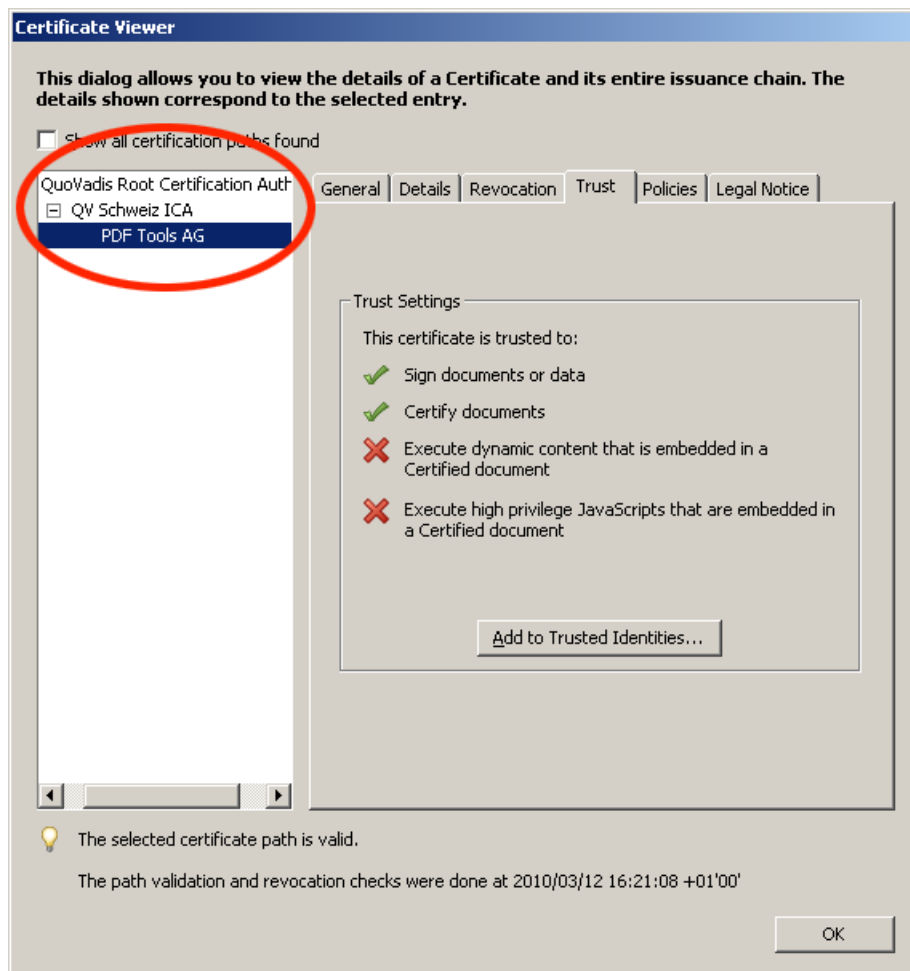
Validation can be in different ways, e.g. Adobe Acrobat, from which the screenshots below are taken.

Trust Chain

Before the trust chain can be validated, ensure the root certificate is trusted. There are different ways to add a certificate as trusted root certificate. The best way on Windows is this:

1. Retrieve a copy of the certificate containing a public key. This can be done by requesting it from the issuer (your CA) or by exporting it from an existing signature to a file (CertExchange .cer). Ensure you are not installing a malicious certificate!
2. Add the certificate to the trusted root certificates. If you have the certificate available as file, you can simply double-click it to install it.

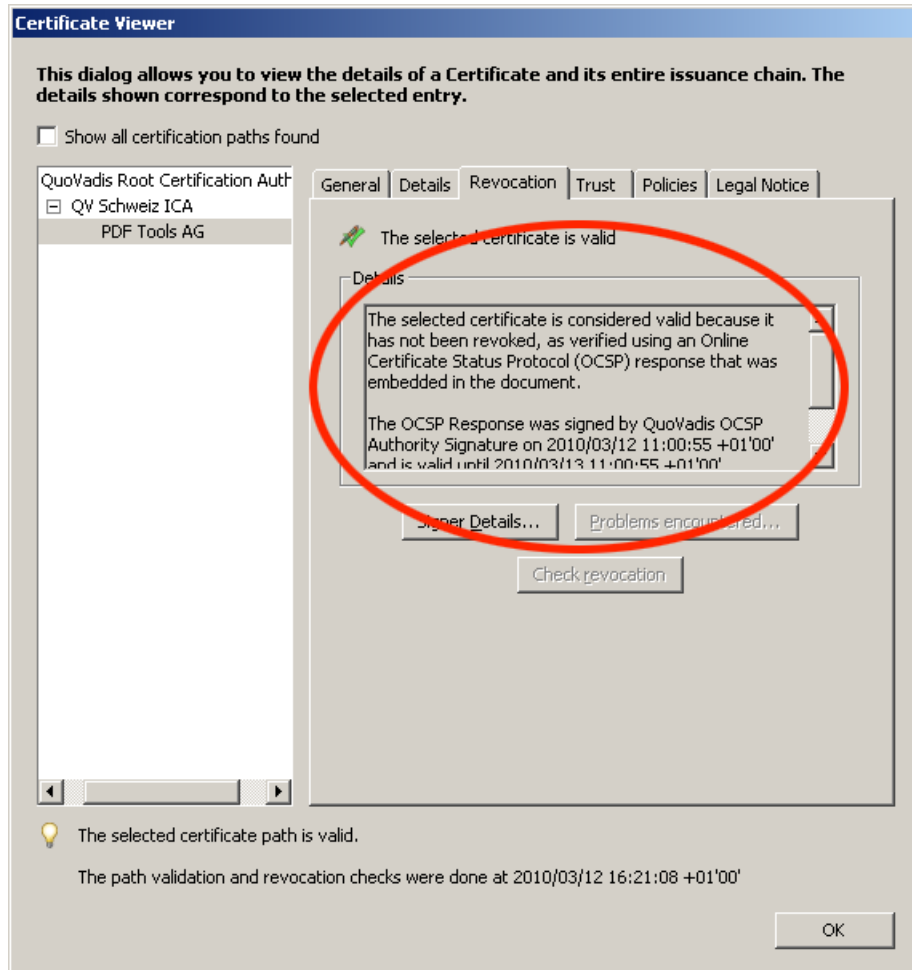
After that you can validate the signature, e.g. by opening the PDF document in Adobe Acrobat, right-click the signature and select "Validate", then select "Properties" and select the tab "Trust". There the certificate should be trusted to "sign documents or data".



Revocation Information

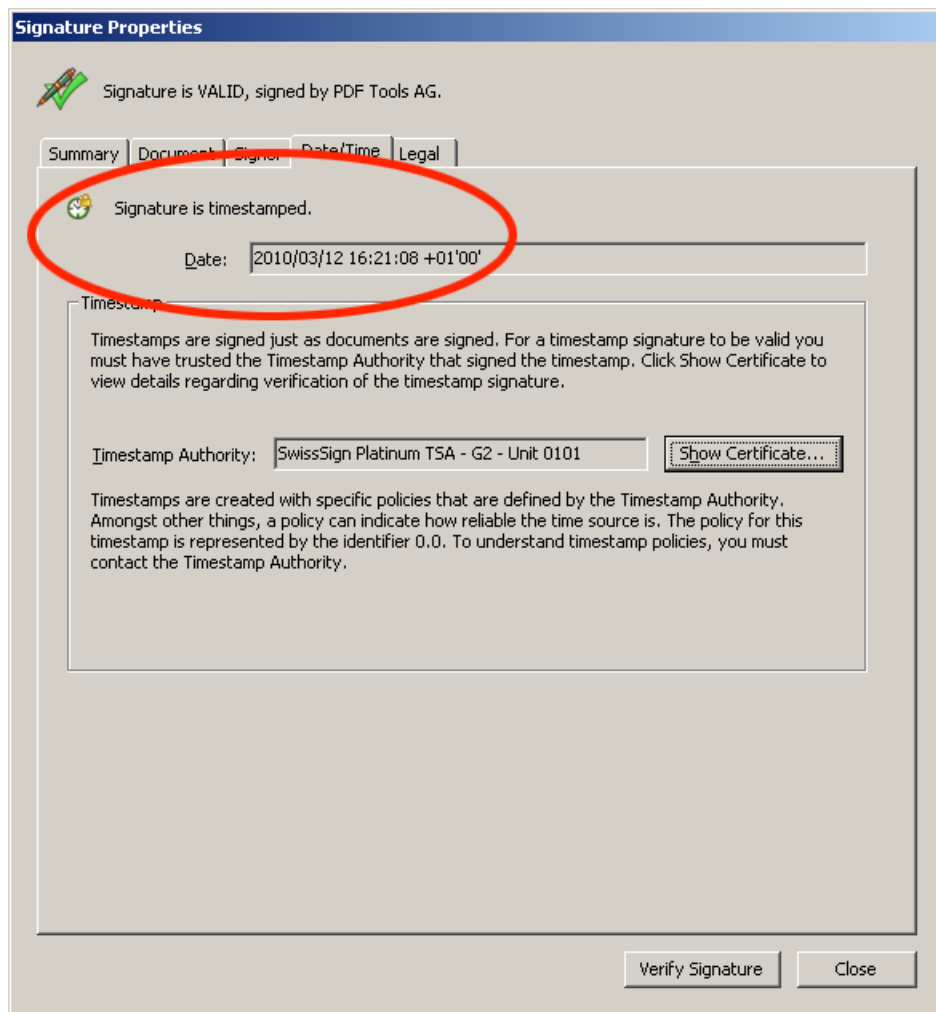
An OCSP response or CRL must be available. This is shown in the tab “Revocation”. The details should mention that “the certificate is considered *valid*”.

The presence of revocation information must be checked for the signing certificate and all certificates of its trust chain except for the root certificate.



Time-stamp

The signature can optionally contain a time-stamp. This is shown in the tab “Date/Time”. The certificate of the time-stamp server must also be trusted, i.e. its trust chain should be validated as described in the section Trust Chain above.



5.7.2 Validation of a PAdES LTV Signature

Verifying if a signature conforms to the PAdES LTV standard is similar to validating a Qualified Electronic Signature.

The following must be checked:

1. Trust Chain
2. Revocation information
3. Time-stamp
4. LTV expiration date
5. Other PAdES Requirements

Trust Chain

Trust chain validation works the same as for validating Qualified Electronic Signatures.

Revocation Information

Revocation information (OCPS response or CRL) must be valid and embedded into the signature. In the details, verify that the revocation check was performed using data that was *“was embedded in the signature or embedded in the document”*. Revocation information that *“was contained in the local cache”* or *“was requested online”* is not embedded into the signature and does not meet PAdES LTV requirements. If Adobe Acrobat claims that revocation

information is contained in the local cache, even though it is embedded into the document, restart Adobe Acrobat and validate the signature again.

Time-stamp

A time-stamp must be embedded and validated as described for validating Qualified Electronic Signatures. If a document contains multiple time-stamps, all but the latest one must contain revocation information.

LTV Expiration Date

The long term validation ability expires with the expiration of the signing certificate of the latest time-stamp.

The life-time of the protection can be further extended beyond the life-of the last time-stamp applied by adding further DSS information to validate the previous last time-stamp along with a new time-stamp. This process is described in chapter [How to Create a PAdES Signature](#).

Other PAdES Requirements

Certain other PAdES requirements, such as requirements on the PKCS#7 CMS, cannot be validated using Adobe Acrobat. For this, use the 3-Heights™ PDF Security API for validation.

5.8 Error Handling

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

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

Example:

```
public Boolean convert(string file, string password, string outfile, string logfile)
{
    using (Pdf2Pdf converter = new Pdf2Pdf())
    {
        [...]
        if (!converter.Convert(file, password, outfile, logfile))
        {
            if (converter.ErrorCode == PDFErrorCode.PDF_E_PASSWORD)
            {
                password = InputBox.Show("Password incorrect. Enter correct password:");
                return convert(file, password, outfile, logfile);
            }
            else if (converter.ErrorCode == PDFErrorCode.PDF_E_CONVERSION)
            {
                return MessageBox.Show(
                    "Please check output file for critical differences.",
                    "Conversion errors occurred.",
                    MessageBoxButtons.YesNo
                ) == DialogResult.Yes;
            }
        }
    }
}
```

```
    }  
    else  
    {  
        MessageBox.Show(String.Format(  
            "Failed to convert file: {0}", converter.ErrorMessage));  
        return false;  
    }  
}  
[...]  
}  
}
```

6 Interface Reference

This chapter lists all available methods and properties of the 3-Heights™ PDF to PDF/A Converter API. The API provides four interfaces: C, Java, .NET and COM. The following documentation is based on the COM interface. The use of the other interfaces and the names of the functions in these interfaces correspond to the COM interface.

6.1 Pdf2Pdf Interface

6.1.1 AddAssociatedFile

Method: Boolean AddAssociatedFile(String FileName, String Name, Integer Associate, String AFRelationship, String MimeType, String Description, DATE ModDate)

Method: Boolean AddAssociatedFileMem(Variant MemBlock, String Name, Integer Associate, String AFRelationship, String MimeType, String Description, DATE ModDate)

Add a file to the document's embedded files in [Convert](#), [ConvertMem](#), [ConvertStream](#). For PDF/A-3, the embedded file is associated with an object of the document, i.e. it is an associated file.

This method must be called before [Convert](#), [ConvertMem](#), [ConvertStream](#). The file is embedded as-is. Embedding files is not allowed for PDF/A-1 and restricted to PDF/A conforming files for PDF/A-2.

Parameters:

FileName [String] The path (or URL) to the file to be embedded.

MemBlock [Variant] The memory buffer containing the file to embed.

Name [String] The name used for the embedded file. This name is presented to the user when viewing the list of embedded files. Default: **FileName** with the path removed.

Associate [Integer] The object to associate the embedded file with. **-1** for none, **0** for document, number greater than **0** for respective page. The default is **0** for PDF/A-3 and **-1** otherwise.

AFRelationship [String] (Default: "Unspecified") The relationship of the embedded file to the object associate. (Ignored, if **Associate** is **-1**.) Allowed values are "Source", "Data", "Alternative", "Supplement" and "Unspecified".

MimeType [String] (Default: "application/octet-stream") Mime-type of the embedded file. Common values other than the default are "application/pdf", "application/xml" or "application/msword".

Description [String] (Default: "") A description of the embedded file. This is presented to the user when viewing the list of embedded files.

ModDate [DATE] The modify date of the file. Default: The modify date of the file on the file system or current time, if not available.

Returns:

True The file was embedded successfully.

False Otherwise.

6.1.2 AddEmbeddedFile

Method: Boolean AddEmbeddedFile(String FileName, String Name)

This is a simplified call that is equal to [AddAssociatedFile](#) with default arguments. This is for convenience, for example when embedding files in a PDF/A-2 conforming document. When embedding files in PDF/A-3, use [AddAssociatedFile](#).

6.1.3 AddFontDirectory

Method: Boolean AddFontDirectory(String Directory)

Fonts must be embedded in order to create a valid PDF/A. If the input file contains a font which is not embedded, the font directory is searched for a font with the same name. If such a font is found, the font is embedded. This method can be used to add (multiple) font directories to the search path for fonts.

Parameter:

Directory [String] The path to the font directory that is to be added to the search path.

Returns:

True The font directory was added successfully.

False Otherwise.

In addition to directories added with this method, the [default font directories](#) are always considered.

See chapter [Fonts](#) for more information on the font directories and font handling of the 3-Heights™ PDF to PDF/A Converter API in general.

6.1.4 AddInvoiceXml

Method: Boolean AddInvoiceXml(TPDFInvoiceType Type, String FileName, String AfRelationship)

Method: Boolean AddInvoiceXmlMem(TPDFInvoiceType Type, Variant MemBlock, String AfRelationship)

Add an XML invoice file (Factur-X or ZUGFeRD).

Note: This feature requires the conformance to be set to PDF/A-3.

If the specified XML invoice file cannot be added during conversion, conversion is aborted with an error `PDF_E_IN-VOICEXML`. This can happen either if the invoice type cannot be determined unambiguously from the XML and there is no clear preference, or if the chosen invoice type is in direct contradiction to the XML itself.

Other than those basic checks, the XML invoice is not validated against any standard or schema.

Parameters:

Type [`TPDFInvoiceType`] The type of invoice.

For the generic values `ePDFInvoiceZugferd` and `ePDFInvoiceFacturX`, the profile is determined automatically. If that's not possible, a profile can be chosen explicitly using the more specific enum values.

FileName [`String`] The path to the XML invoice file.

MemBlock [`Variant`] The XML invoice file as a memory buffer.

AFRelationship [`String`] Optional AFRelationship value.

The AFRelationship determines the relation of the invoice file to the PDF. Allowed values are `"Source"`, `"Data"`, `"Alternative"`, `"Supplement"` and `"Unspecified"`.

Note that some invoice standards restrict the set of allowed values for certain profiles.

Returns:

True The file was found.

False Otherwise.

6.1.5 AddSignature

Method: Boolean `AddSignature(PdfSignature pSignature)`

License feature: `Signature`

Add a digital signature to the document. The signature is defined using a `PdfSignature` object. This method must be called prior to `Convert`, `ConvertMem`, `ConvertStream`. Do not dispose of the `PdfSignature` object until the associated document has been converted.

More information on applying digital signatures can be found in Chapter [How to Create Electronic Signatures](#).

Parameter:

pSignature [`PdfSignature`] The digital signature that is to be added. The properties of the signature must be set before it is added.

Returns:

True Successfully added the signature to the document.

Note: At this point it is not verified whether the certificate is valid or not. If an invalid certificate is provided the [Convert](#), [ConvertMem](#), [ConvertStream](#) function will fail later on.

False Otherwise.

6.1.6 AddZUGFeRDXml

[Deprecated] Method: Boolean [AddZUGFeRDXml](#)(String FileName)

[Deprecated] Method: Boolean [AddZUGFeRDXmlMem](#)(Variant MemBlock)

These methods were deprecated in version 5.6, use methods [AddInvoiceXml](#) and [AddInvoiceXml](#) instead.

6.1.7 AllowDowngrade

Property (get, set): Boolean [AllowDowngrade](#)

Default: **False**

If set to **True**, automatic downgrade of the PDF/A conformance level is allowed, e.g. from PDF/A-1a to PDF/A-1b.

If this property is set to **True**, the level is downgraded under the following conditions:

- Downgrade to level B: If a file contains text that is not extractable (i.e. missing ToUnicode information).
Example: Downgrade PDF/A-2u to PDF/A-2b.
- Downgrade to level U (PDF/A-2 and PDF/A-3) or B (PDF/A-1): Level A requires logical structure information and “tagging” information, so if a file contains no such information, its level is downgraded.
Logical structure information in a PDF defines the structure of content, such as titles, paragraphs, figures, reading order, tables or articles. Logical structure elements can be “tagged” with descriptions or alternative text. “Tagging” allows the contents of an image to be described to the visually impaired.
It is not possible for the 3-Heights™ PDF to PDF/A Converter API to add meaningful tagging information. Adding tagging information without prior knowledge about the input file’s structure and content is neither possible nor allowed by the PDF/A standard. For that reason, the conformance level is automatically downgraded to level B or U.
Example: Downgrade PDF/A-1a to PDF/A-1b.

If set to **False** and an input file cannot be converted to the requested standard, e.g. because of missing “tagging” information, the conversion is aborted and the [ErrorCode](#) set to [PDF_E_DOWNGRADE](#).

6.1.8 AllowUpgrade

Property (get, set): Boolean [AllowUpgrade](#)

Default: **False**

If set to **True**, automatic upgrade of the PDF/A version is allowed. If the target standard is PDF/A-1 and a file contains elements that cannot be converted to PDF/A-1, the target standard is upgraded to PDF/A-2. This avoids significant visual differences in the output file.

For example, the following elements may lead to an automatic upgrade:

- Transparency
- Optional content groups (OCG, layers)
- Real values that exceed the implementation limit of PDF/A-1
- Embedded OpenType font files
- Predefined CMap encodings in Type0 fonts

If set to **False**, the conformance is not upgraded. Depending on the value of the [ConversionErrorMask](#) the conversion this will fail with a conversion error [PDF_E_CONVERSION](#).

6.1.9 AnalyzeOnly

Property (get, set): Boolean [AnalyzeOnly](#)
Default: **False**

When set to true, the method [Convert](#), [ConvertMem](#), [ConvertStream](#) analyzes the input file only, it does not create an output file. The results of the analysis are written to the corresponding log file. If [Convert](#), [ConvertMem](#), [ConvertStream](#) returns True, the file conforms to the compliance set in [Compliance](#). If issues are found, False is returned and the [ErrorCode](#) is set to [PDF_E_CONFORMANCE](#).

The analysis is similar to the analysis using the 3-Heights™ PDF Validator. However, the 3-Heights™ PDF to PDF/A Converter API is more strict in certain issues, especially concerning those corner cases of the PDF/A ISO Standard in which a conversion is strongly advised.

6.1.10 BeginSession

Method: Boolean [BeginSession](#)(String [Provider](#))

The methods [BeginSession](#) and [EndSession](#) support bulk digital signing by keeping the session to the security device (HSM, Token or Cryptographic Provider) open. See the Section [Guidelines for Mass Signing](#) for more guidelines.

For backwards compatibility the use of these methods is optional. If used, the [Provider](#) property may not be set. If omitted, an individual session to the provider indicated by the property [Provider](#) is used for each signature operation.

Parameter:

Provider [String] See property [Provider](#).

Returns:

True Session started successfully.

False Otherwise.

6.1.11 ColorSpaceProfile

Property (get, set): String ColorSpaceProfile
Default: "%SystemRoot%\System32\spool\drivers\color\USWebCoatedSWOP.icc" (Windows only)

This property is used to set and get ICC color profile file names. ICC profiles can be set prior to [Convert, ConvertMem, ConvertStream](#) and read after [Convert, ConvertMem, ConvertStream](#).

Setting ICC profiles makes the converter substitute device color spaces with ICC based color spaces. At most three ICC profiles can be set, as substitutes for DeviceRGB, DeviceCMYK, and DeviceGray respectively. To set several ICC profiles, set this property several times. The matching device color space to be substituted is selected automatically. Note that it is not necessary to set ICC profiles for a successful conversion, see section [Color Spaces](#).

When getting this property, only one ICC profile file name can be queried: If the output intent is set to an RGB profile (a CMYK profile) then the file name of any set CMYK profile (RGB profile respectively) results. Otherwise any set CMYK or RGB profile results with CMYK taking precedence.

If a required color space profile is not available, a default color space is generated.

C and Java interfaces only: If the provided path is not a valid profile, the method fails.

6.1.12 Compliance

Property (get, set): TPDFCompliance Compliance
Default: ePDFa2b

This property sets or gets the conformance level of the output PDF. Supported values for the enumeration [TPDF-Compliance](#) are:

- | | |
|-----------|-----------|
| ■ ePDFa1a | ■ ePDFa2u |
| ■ ePDFa1b | ■ ePDFa3a |
| ■ ePDFa2a | ■ ePDFa3b |
| ■ ePDFa2b | ■ ePDFa3u |

Other listed entries (e.g. [ePDF10](#), [ePDF11](#), ...[ePDF17](#), [ePDFUnk](#)) are not supported as output conformance level by the 3-Heights™ PDF to PDF/A Converter API.

Some files cannot be converted to the conformance requested. The 3-Heights™ PDF to PDF/A Converter API can detect this and up- ([AllowUpgrade](#)) or downgrade ([AllowDowngrade](#)) the conformance automatically.

6.1.13 ConversionErrors

Property (get): TPDFConversionError ConversionErrors

Get conversion error events that occurred during conversion. This property should be queried after [Convert, ConvertMem, ConvertStream](#) returned `False` and the property [ErrorCode](#) is set to `PDF_E_CONVERSION`.

See chapter [Conversion Errors](#) for more information on conversion errors and how they can be handled.

6.1.14 ConversionErrorMask

Property (get, set): TPDFConversionError ConversionErrorMask
Default: ePDFConversionErrorVisualDiff + ePDFConversionErrorOCGRemoved + ePDFConversionErrorTranspRemoved + ePDFConversionErrorEFRemoved + ePDFConversionErrorXMPRemoved + ePDFConversionErrorCorrupt

Define which conversion operations shall result in a conversion error, i.e. [Convert](#), [ConvertMem](#), [ConvertStream](#) return **False** and the property [ErrorCode](#) is set to [PDF_E_CONVERSION](#). In the case of a conversion error, use the property [ConversionErrors](#) to retrieve the actual conversion error events that occurred during the conversion.

See enumeration [TPDFConversionError](#) for a list of supported conversion error events.

See chapter [Conversion Errors](#) for more information on conversion errors and how they can be handled.

Example: In order to accept the removal of XMP metadata, remove [ePDFConversionErrorXMPRemoved](#) from [ConversionErrorMask](#).

```
ConversionErrorMask = ConversionErrorMask And Not ePDFConversionErrorXMPRemoved
```

6.1.15 Convert, ConvertMem, ConvertStream

Method: Boolean [Convert](#)(String InputFileName, String Password, String OutputFileName, String LogFileName)
Method: Boolean [ConvertMem](#)(Variant InputBytes, String Password, Variant* OutputBytes, Variant* LogBytes)
Method: Boolean [ConvertStream](#)(Variant InputStream, String Password, Variant OutputStream, Variant LogStream)

Open a PDF document convert it to PDF/A and save the output document.

An overview of the conversion process is provided in section [Process Description](#).

Parameters:

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

Password [String] (optional) The user or the owner password of the encrypted input PDF document. If this parameter is left out an empty string is used as a default.

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

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

InputBytes [Variant] A byte array containing the input PDF document.

OutputBytes [Variant*] A byte array containing the output PDF/A document. When using the [C](#) interface, the returned byte array must be freed using [Pdf2PdfFreeMem\(\)](#).

LogBytes [Variant*] A byte array containing the log. When using the [C](#) interface, the returned byte array must be freed using [Pdf2PdfFreeMem\(\)](#).

InputStream [Variant] A readable stream containing the input PDF document.

OutputStream [Variant] A stream to which the output PDF/A document is written. This stream must be readable and writable.

LogStream [Variant] A stream to which the log is written.

Returns:

True If the function was executed successfully, i.e. could read and convert the input file.

False If no valid output document is written. Check [ErrorCode](#) and [ErrorMessage](#) to get the cause of the problem. Possible causes are:

- License is not set or invalid
- Input file does not exist
- Input file is protected by a user password and the provided password is incorrect
- Output file is not writable, e.g. locked
- Conversion was stopped
- The input file is not a PDF file or contains unrendered XFA fields.
- The post analysis detected an error.
- A conversion error occurred.

6.1.16 ConvertAlways

Property (get, set): Boolean [ConvertAlways](#)
Default: **False**

Setting the property [ConvertAlways](#) to true forces the conversion even if the input file already conforms to the requested standard.

6.1.17 EmbedAllFonts

Property (get, set): Boolean [EmbedAllFonts](#)
Default: **False**

By default, fonts are not embedded unless required to achieve PDF/A conformance. For example, fonts of OCR text (invisible text not used for rendering) do not have to be embedded. Setting this property to **True** forces all fonts to be embedded. This produces larger PDF/A output files and is intended only as a workaround for bugs in subsequent systems.

6.1.18 EmbedT1asCFF

Property (get, set): Boolean `EmbedT1asCFF`
Default: `False`

Convert Type1 (PostScript) fonts to Compact Font Format before embedding. This reduces the file size. This affects the embedding of fonts only, existing Type1 fonts of the input document will not be converted.

6.1.19 EndSession

Method: Boolean `EndSession()`

Ends the open session to the security device.

See [BeginSession](#).

6.1.20 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 to PDF/A Converter API has returned a value, which signals a failure of the function (see chapter [Error Handling](#)). See also enumeration [TPDFErrorCode](#). PDF-Tools error codes are listed in the header file `bseerror.h`. Please note that only few of them are relevant for the 3-Heights™ PDF to PDF/A Converter API.

6.1.21 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 to PDF/A Converter API has returned a value, which signals a failure of the function (see chapter [Error Handling](#))

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

6.1.22 ExportText

Method: Boolean `ExportText(String FileName)`

Export the retrieved OCR text to a file. This function can only be used in combination with an OCR engine (see method [SetOCREngine](#)). When an OCR engine is set, the OCR text is always embedded in the resulting PDF document. If this method is used, it is in addition also extracted to a file.

The output format is a table, where rows are separated by a new line and columns are separated by a tabulator.

The table contains the following columns:

Columns	Description
Page	Page number
Image	PDF object number which contains the image
FontSize	Font size in points
FontName	Font name, for any barcode font the name is "Barcode". This value is only set if the font name is returned by the OCR engine.
FontFamily	1 Serif 2 SansSerif 3 Monospaced This value is only set if provided by the OCR engine.
FontStyles	1 Barcode 2 Bold 4 Italic 8 Underline 16 Strikeout This value is only set if provided by the OCR engine. Example: 6 = 2 + 4 = Bold + Italic
Baseline	Baseline of the text
Left, Top, Right, Bottom	Bounding box of the text in PDF coordinates
String	Recognized text

6.1.23 FlattenSignatures

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

Remove all signed signature fields and add their appearances to the page's content. Note that the signatures themselves (the cryptographic parts) are removed and hence the bit `ePDFConversionErrorDocSigned` of the conversion error is set regardless of the value of the `FlattenSignatures` property.

Processing the PDF with 3-Heights™ PDF to PDF/A Converter API breaks existing signatures and their cryptographic parts need to be removed. In general, the visual appearances of signatures are regarded as worthless without the

cryptographic part and are therefore removed by default as well. The visual appearances can be preserved by setting this property [FlattenSignatures](#) to [True](#).

6.1.24 ForceEmbeddingOfCMaps

Property (get, set): Boolean [ForceEmbeddingOfCMaps](#)
Default: [False](#)

Set this option to [True](#), to force the embedding of all predefined CMaps (encoding of composite fonts). By default, predefined CMaps are embedded by the 3-Heights™ PDF to PDF/A Converter API if required to achieve PDF/A conformance. Therefore, the value of this option has no effect on the PDF/A conformance of the output document.

6.1.25 GetOCRPluginCount

Method: Integer [GetOCRPluginCount\(\)](#)

OCR engines are accessed through the corresponding OCR interface DLLs. At present the following OCR engines are supported:

Abbyy FineReader 11 OCR Engine This engine is accessed by the OCR interface DLL `pdfocrpluginAbbyy11.ocr`.

Abbyy FineReader 10 OCR Engine This engine is accessed by the OCR interface DLL `pdfocrpluginAbbyy10.ocr`.

3-Heights™ OCR Service This service is accessed by the OCR interface DLL `pdfocrpluginService.ocr`. The service accesses the Abbyy FineReader 10 or 11 OCR Engine.

The OCR interface DLL is provided by the 3-Heights™ PDF to PDF/A Converter API.

The OCR engine is provided as a separate product: 3-Heights™ OCR Enterprise Add-On.

In order to make use of the OCR engine, the OCR interface DLL and the OCR engine must be installed. The property [GetOCRPluginCount](#) returns the number of available OCR interface DLLs. It does not verify the corresponding OCR engines are installed and can be initialized. The OCR engine is loaded with the method [SetOCREngine](#).

Returns:

The number of available OCR engines (i.e. their corresponding OCR interface DLLs).

6.1.26 GetOCRPluginName

Method: String [GetOCRPluginName\(Integer iOCREngine\)](#)

An OCR engine is accessed through an OCR plug-in. Each plug-in corresponds to one OCR engine. The number of OCR plug-ins is retrieved using [GetOCRPluginCount](#). The method call [GetOCRPluginName\(n\)](#) returns the

name of the nth OCR Engine which corresponds to that OCR plug-in. At present there are three OCR engines available: "abbyy11", "abbyy10" and "service".

Parameter:

iOCREngine [Integer] The number of the OCR engine. The total number of engines is retrieved using [GetOCRPluginCount](#).

Returns:

The name of the nth OCR engine. **Nothing** if it does not exist.

6.1.27 ImageQuality

Property (get, set): Integer ImageQuality
Default: 80

Set or get the image quality index for images that use a prohibited lossy compression type and must be recompressed.

Supported values are **1** to **100**. A higher value means better visual quality at the cost of a larger file size. Recommended values range from **70** to **90**.

Example:

JPX (JPEG2000) is not allowed in PDF/A-1. If a PDF contains a JPX compressed image, its compression type must be altered. Thus the 3-Heights™ PDF to PDF/A Converter API converts it to an image with JPEG compression using the image quality defined by this property.

6.1.28 InfoEntry

Method: String InfoEntry (String Key)

Retrieve or add a key-value pair to the document info dictionary. Values of predefined keys are also stored in the XMP metadata package.

Popular entries specified in the [PDF Reference 1.7](#) and accepted by most PDF viewers are "Title", "Author", "Subject", "Creator" (sometimes referred to as Application) and "Producer" (sometimes referred to as PDF Creator).

Parameter:

Key [String] A key as string.

Returns:

The value as string.

Note: Note that the getter does not return values of the input document but merely those that have previously been set using [InfoEntry](#).

Examples in Visual Basic 6:

Set the document title.

```
conv.InfoEntry("Title") = "My Title"
```

Set the creation date to 13:55:33, April 5, 2010, UTC+2.

```
conv.InfoEntry("CreationDate") = "D:20100405135533 + 02'00'"
```

6.1.29 LicenseIsValid

Property (get): Boolean `LicenseIsValid`

Check if the license is valid.

6.1.30 Linearize

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

Get or set whether to linearize the PDF output file, i.e. optimize file for fast web access.

The 3-Heights™ PDF to PDF/A Converter API does not support linearization of PDF 2.0 documents. For such documents, processing fails.

A linearized document has a slightly larger file size than a non-linearized file and provides the following main features:

- When a document is opened in a PDF viewer of a web browser, the first page can be viewed without downloading the entire PDF file. In contrast, a non-linearized PDF file must be downloaded completely before the first page can be displayed.
- When another page is requested by the user, that page is displayed as quickly as possible and incrementally as data arrives, without downloading the entire PDF file.

The above applies only if the PDF viewer supports fast viewing of linearized PDFs.

The 3-Heights™ PDF to PDF/A Converter API cannot linearize signed files. So this property must be set to `False` if a digital signature is applied.

When enabling this option, then no PDF objects will be stored in object streams in the output PDF. For certain input documents this can lead to a significant increase of file size.

6.1.31 NoCache

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

Get or set whether to disable the cache for CRL and OCSP responses.

Using the cache is safe, since the responses are cached as long as they are valid only. The option affects both signature creation and validation.

See section on [Caching of CRLs, OCSP, and Time-stamp Responses](#) for more information on the caches.

6.1.32 NoDSS

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

Set this option to **True** to not embed revocation information (OCSP, CRL, and trust chain) in the document security store (DSS) when signing documents. Use this option to work around issues with legacy software that does not support the DSS. The use of the DSS is recommended for long-term (LTV) signatures.

6.1.33 OCRBitonalRecognition

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

Specify whether the images should be converted to bi-tonal (black and white) before OCR recognition.

Enabling this feature can improve the memory consumption of the OCR process.

Enabling this feature automatically re-embeds the original images in the output document. The setting of the property [OCRReembedImages](#) is therefore ignored.

6.1.34 OCRDeskewImage

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

Correct the skew angle of images.

This option set to **True** has only an effect if the required information is provided by the OCR engine, which depends on the type and settings of the engine.

This option set to **True** might change the appearance of the page and is only recommended for simple scanned documents that consist of a single image.

Using the option for digital-born documents may destroy the page layout.

6.1.35 OCREmbedBarcodes

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

This property specifies whether the recognized barcodes are embedded in the XMP metadata.

6.1.36 OCRReembedImages

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

This option set to `True` currently requires the [OCRDeskewImage](#) to be also set to `True`.

The OCR engine de-skews and de-noises the input image before recognizing the characters. This option controls whether the 3-Heights™ PDF to PDF/A Converter API should use the preprocessed image or keep the original image.

Setting this option to `True` has only an effect if the preprocessed image is provided by the OCR engine, which depends on the type and settings of the engine.

If this option is set to `True`, the resulting image may have a different color space, compression and size.

Since this option currently requires [OCRDeskewImage](#), it is recommended only for simple scanned documents.

6.1.37 OCRMode

Property (get, set): Integer `OCRMode`
Default: `1`

Specify behavior of the converter for files with existing OCR text. Available OCR modes are the following:

OCR mode	Description
1	Only perform OCR for images without existing OCR text (default).
2	If OCR engine is active, remove old OCR text and perform OCR for all images. Hence, existing OCR text is not removed, if OCR engine is not active.
3	Always remove old OCR text and, if OCR engine is active, perform OCR for all images. This can be used to strip existing, without adding new OCR text.
4	Only perform OCR if the input file contains no text.

6.1.38 OCRResolutionDPI

Property (get, set): Single `OCRResolutionDPI`
Default: `300`

Resample images to target resolution before they are sent to the OCR engine. The default is **300** DPI, which is the preferred resolution for most OCR engines.

6.1.39 OCRRotatePage

Property (get, set): Boolean `OCRRotatePage`
Default: **False**

This property specifies whether the page is rotated according to the recognized image rotation.

6.1.40 OCRThresholdDPI

Property (get, set): Single `OCRThresholdDPI`
Default: **400**

Only images with a higher resolution than the threshold are re-sampled before OCR. The default is **400** DPI. If set to **-1**, no re-sampling is applied.

6.1.41 OutputIntentProfile

Property (get, set): String `OutputIntentProfile`
Default: **"%SystemRoot%\System32\spool\drivers\color\USWebCoatedSWOP.icc"**
(Windows only)

Set or get the path to the ICC profile for the output intent. This property can be set prior to [Convert](#), [ConvertMem](#), [ConvertStream](#) and can be read after [Convert](#), [ConvertMem](#), [ConvertStream](#). See section [Color Spaces](#) for the usage of the output intent.

It is not recommended to set an output intent using this property. Instead, the ICC profiles for device-specific color spaces should be set using the property [ColorSpaceProfile](#), which ensures an optimal result of the automatic color conversion algorithm (see [PDF/A Requirements](#)).

If an invalid path is provided, [Convert](#), [ConvertMem](#), [ConvertStream](#) fails and writes a corresponding message to the log file.

The given profile is embedded only if the input file does not contain a PDF/A output intent already.

If during conversion an output intent was set automatically, then the path for this profile can be queried after conversion by getting this property.

C and Java interface only: If the provided path is not a valid profile, the method fails.

6.1.42 PostAnalyze

Property (get, set): Boolean `PostAnalyze`
Default: **True**

Analyze the created PDF output file and verify if it meets the specified conformance level. The result of this analysis is written to the log file. If the post analysis detects an error, [Convert](#), [ConvertMem](#), [ConvertStream](#) returns **False** and the [ErrorCode](#) is [PDF_E_POSTANALYSIS](#).

The post analysis is executed only if an output file was created and the conversion was successful. The property [PostAnalysis](#) is ignored if the property [AnalyzeOnly](#) is true.

The post analysis can detect errors in the created output file that could not be predicted based on the analysis of the input file nor could they be detected during the conversion, because the conversion also depends on the input parameters (such as ICC profiles).

The post-analysis is equal to the analysis using the 3-Heights™ PDF Validator and validating against PDF/A.

6.1.43 ProductVersion

Property (get): String [ProductVersion](#)

Get the version of the 3-Heights™ PDF to PDF/A Converter API in the format "A.C.D.E".

6.1.44 RemoveSignature

[Deprecated] Property (get, set): Boolean [RemoveSignature](#)

Default: **True**

This property is deprecated, instead use [ePDFConversionErrorDocSigned](#), see [ConversionErrorMask](#).

6.1.45 ReportDetails

Property (get, set): Boolean [ReportDetails](#)

Default: **False**

Write a detailed list of errors and warnings from the analysis of the input file as well as the post-analysis of the output file to the log file.

Setting this property to true, the conversion step lists all violations per page. Each violation is listed with a page number (page 0 = document level), error number, a description and a counter of how many times the error occurs. The option provides more detailed information than [ReportSummary](#). All errors are listed in the header file `bseerror.h`.

Examples of possible errors:

```
0, 0x80410604, "The key Metadata is required but missing.", 1
0, 0x80410604, "The key MarkInfo is required but missing.", 1
1, 0x00418704, "The font Arial-BoldMT must be embedded.", 1
1, 0x83410612, "The document does not conform to the requested standard.", 1
```


6.1.46 ReportSummary

Property (get, set): Boolean `ReportSummary`

Default: `False`

Write a summary of errors and warnings from the analysis of the input file as well as the post-analysis of the output file to the log file. If any of the following violations is detected at least once, it is reported (once). This report provides less detailed information than the detailed list per page provided by [ReportDetails](#).

- The file format (header, trailer, objects, xref, streams) is corrupted.
- The document doesn't conform to the PDF reference (missing required entries, wrong value types, etc.).
- The file is encrypted and the password was not provided.
- The document contains device-specific color spaces.
- The document contains illegal rendering hints (unknown intents, interpolation, transfer and halftone functions)
- The document contains alternate information (images).
- The document contains embedded PostScript code.
- The document contains references to external content (reference XObjects, file attachments, OPI).
- The document contains fonts without embedded font programs or encoding information (CMAPs).
- The document contains fonts without appropriate character to Unicode mapping information (ToUnicode maps).
- The document contains transparency.
- The document contains unknown annotation types.
- The document contains multimedia annotations (sound, movies).
- The document contains hidden, invisible, non-viewable or non-printable annotations.
- The document contains annotations or form fields with ambiguous or without appropriate appearances.
- The document contains actions types other than for navigation (launch, JavaScript, ResetForm, etc.).
- The document's meta data is either missing or inconsistent or corrupt.
- The document doesn't provide appropriate logical structure information.
- The document contains optional content (layers).

6.1.47 SetLicenseKey

Method: Boolean `SetLicenseKey(String LicenseKey)`

Set the license key.

6.1.48 SetMetadata

Method: Boolean `SetMetadata(String FileName)`

Set the document's XMP metadata. The XMP metadata is inserted as is, which means it is not parsed and validated. If no XMP metadata is provided, the 3-Heights™ PDF to PDF/A Converter API generates it automatically.

Parameter:

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

Returns:

True The XMP metadata file was set successfully.

False Otherwise.

6.1.49 SetOCREngine

Method: Boolean `SetOCREngine(String Engine)`

This method requires the 3-Heights™ OCR Add-On, which is a separate product, to be installed. See also documentation for the 3-Heights™ OCR Add-On.

Set the OCR engine that is used when OCR information shall be added during the conversion. If the engine's name is set to an empty string, OCR is not applied.

Parameter:

Engine [String] The name of the OCR engine (e.g. "abbyy11"). For every available OCR engine, there is a corresponding OCR interface DLL. The OCR interface DLLs (e.g. pdfocrAbbyy11.ocr) are distributed with the 3-Heights™ PDF to PDF/A Converter API and are required to communicate with the OCR engine. The names of all available OCR engines can be retrieved using the properties [GetOCRPluginCount](#) and [GetOCRPluginName](#).

Returns:

True The OCR interface DLL was found, the OCR engine was found and the OCR engine was successfully initialized.

False Otherwise.

6.1.50 SetOCRLanguages

Method: Boolean `SetOCRLanguages(String Languages)`

This method requires the 3-Heights™ OCR Add-On, which is a separate product, to be installed. See also documentation for the 3-Heights™ OCR Add-On.

Setting languages helps the OCR engine to minimize errors by means of using dictionaries of the defined languages.

This method must be called after [SetOCREngine](#).

If [SetOCRParams](#) is used, [SetOCRLanguages](#) must be called after [SetOCRParams](#).

Parameter:

Languages [String] A string of one or multiple, comma-separated languages. The supported names depend on the OCR engine. The OCR engine will only use dictionaries of the set languages.

Returns:

True The language(s) were successfully set.

False Otherwise.

Example:

```
SetOCREngine("abbyy11")
SetOCRLanguages("English, German")
```

6.1.51 SetOCRParams

Method: Boolean SetOCRParams(String Params)

This method requires the 3-Heights™ OCR Add-On, which is a separate product, to be installed. See also documentation for the 3-Heights™ OCR Add-On.

By means of this method, OCR engine specific settings can be applied in the form of key-value pairs. These pairs depend on the OCR engine and are described in the corresponding manual.

Parameter:

Params [String] A list of comma-separated key value pairs. See example.

Returns:

True The OCR parameters were successfully set.

False Otherwise.

Example: Set a predefined profile for ABBYY 11.

```
SetOCREngine("abbyy11")
SetOCRParams("PredefinedProfile = DocumentArchiving_Accuracy")
```

6.1.52 SetSessionProperty

Method: Boolean SetSessionPropertyString(String Name, String Value)

Method: Boolean SetSessionPropertyBytes(String Name, Variant Value)

Provider-specific session configuration.

Properties have to be set before calling [BeginSession](#) and are deleted when calling [EndSession](#).

Parameters:

Name [String] The name of the property. The names that are supported are specific to the provider used with [BeginSession](#).

Value [String] The value of the property as string.

Value [Variant] The value of the property as byte array.

6.1.53 SetToUnicodeFile

Method: Void SetToUnicodeFile(String FileName)

Update the fonts' Unicodes as specified by file. The file must contain the mapping of character codes to Unicodes for specific fonts.

Note: Note that the code to Unicode mapping depends on the exact font used, its version as well as the way it is embedded into the PDF. All of these parameters, including the name of the embedded font, can freely be chosen by the application that creates a PDF.

One cannot assume that a mapping that is correct for one file is also correct for another. Therefore, the same ToUnicode mapping file can only be applied to files originating from the exact same process.

Example: The following file `unicodes.ini` maps the character code 262 to the ligature “fi” with the Unicode U+FB01.

```
[AppliedSansPro-Light]
262=0xFB01
```

A possible method to determine the ToUnicode mapping file is the following:

1. The conversion must be performed with [ReportDetails](#) set to **True**.
2. The Pre-Analysis will contain messages regarding missing Unicodes:

```
"input.pdf", 1, 38, 0x00418623, "The Unicode for cid 262 is unknown.", 1
```

This message indicates the cid (character code), the object number (in 3rd column) indicates the font, e.g. in this example cid 262 of font object 38.

3. The text of `input.pdf` can be extracted using the `pdftxt`, which is part of the product 3-Heights™ PDF Extract Shell:

```
pdftxt -u -h -oo -o text.txt input.pdf
```

4. In `text.txt` it can be seen, that the name of the font with object number 38 is “ORCNO+AppliedSansPro-Light”:

```
1,161,596,127,"ORCNO+AppliedSansPro-Light",7.0,38,22,"the Asia-Paci•c Region"
```

For the mapping in `unicodes.ini`, the subset prefix (six characters followed by “+”) must be removed, which produces “AppliedSansPro-Light”.

5. `text.txt` must be searched for text of the font “ORCNO+AppliedSansPro-Light” that could not successfully be extracted, e.g. “Paci•c”. It could be assumed, that cid 262 corresponds to the missing character “fi”, which corresponds the Unicode U+FB01.

6. The missing Unicode must be added to `unicodes.ini` and the conversion performed again. The text extracted from the converted file should now be correct. Otherwise above assumption was wrong and must be changed.
7. These steps must be repeated until all text can be extracted successfully and no more post-analysis errors occur.

6.1.54 SubsetFonts

Property (get, set): Boolean `SubsetFonts`
Default: `True`

By default, fonts that are embedded are automatically subset to minimize the file size. If for any reason, e.g. post-processing, fonts shall not be subset, set the property `SubsetFonts` to `False`. Whether fonts are subset or not is irrelevant with respect to the conformance to PDF/A. (Relevant is only that all used glyphs are contained in the font program.)

If this property is set to `False`, embedded fonts, that are subsetted, are replaced with non-subsetted fonts from the local system. To avoid visual differences, embedded fonts are only replaced if they match the system font. Therefore setting `SubsetFonts` to `False` is not recommended, unless it can be guaranteed that all fonts match.

6.1.55 Terminate

Method: Void `Terminate()`

Terminate all open sessions, and finalize and unload all PKCS#11 drivers. Calling `Terminate` is mandatory, if a PKCS#11 device is used for signature creation or validation (see [PKCS#11 Provider](#)). Some drivers require `Terminate` to be called. Otherwise, your application might crash and/or your HSM, USB token, or smart card might not be unlocked.

Make sure to end all open sessions and dispose of all Pdf2Pdf 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.

6.1.56 TestSession

Method: Boolean `TestSession()`

Test if the current session is still alive.

Returns:

True Subsequent calls to [Convert](#), [ConvertMem](#), [ConvertStream](#) are likely to succeed.

False Subsequent calls to [Convert](#), [ConvertMem](#), [ConvertStream](#) are unlikely to succeed. Error codes are the same as in [Convert](#), [ConvertMem](#), [ConvertStream](#) where applicable.

6.1.57 TryConvertEmbPDF

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

By default, embedded files are copied as-is during conversion to PDF/A-3. If TryConvertEmbPDF is set to True, the 3-Heights™ PDF to PDF/A Converter API tries to convert embedded PDF documents to PDF/A. The converted document is embedded only, if the conversion was successful.

Note that this property is relevant for PDF/A-3 only. During conversion to PDF/A-1, all embedded files are removed, whereas the conversion to PDF/A-2 converts all embedded files to PDF/A.

6.2 PdfSignature Interface

This interface allows creating a signature and setting its position and appearance. The visual part of the signature consists of two (multi-line) texts. The string of both texts are generated automatically based on the signature properties if not set manually.

6.2.1 ContactInfo

Property (get, set): String ContactInfo
Default: ""

Add a descriptive text as signer contact info, e.g. a phone number. This enables a recipient to contact the signer to verify the signature. This is not required in order to create a valid signature.

If this property is set to an empty string, no entry is created.

6.2.2 EmbedRevocationInfo

Property (get, set): Boolean EmbedRevocationInfo
Default: True

Embed revocation information such as online certificate status response (OCSP - RFC 2560) and certificate revocation lists (CRL - RFC 3280).

Revocation information of a certificate is provided by a validation service at the time of signing and acts as proof that at the time of signing the certificate is valid. This is useful because even when the certificate expires or is revoked at a later time, the signature in the signed document remains valid.

Embedding revocation information is optional but suggested when applying advanced or qualified electronic signatures.

This property is not supported by all cryptographic providers and never for document time-stamp signatures.

Revocation information is embedded for the signing certificate and all certificates of its trust chain. This implies that both OCSP responses and CRLs can be present in the same message.

The downsides of embedding revocation information are the increase of the file size (normally by around 20 KB) and that it requires a web request to a validation service, which delays the process of signing. For mass signing it is suggested to use the caching mechanism, see chapter [Caching of CRLs, OCSP, and Time-stamp Responses](#).

Embedding revocation information requires an online connection to the CA that issues them. The firewall must be configured accordingly. In case a [web proxy](#) is used, it must be ensured the following MIME types are supported when using OCSP (not required for CRL):

[application/ocsp-request](#)

[application/ocsp-response](#)

If [EmbedRevocationInfo](#) is set to **True**, but the embedding failed, e.g. because the OCSP server is not reachable, the return value of [Convert](#), [ConvertMem](#), [ConvertStream](#) is **False**, and the [ErrorCode](#) after [Convert](#), [ConvertMem](#), [ConvertStream](#) is [SIG_CREA_E_OCSP](#).

6.2.3 FillColor

Property (get, set): Long [FillColor](#)
Default: **16761024** (**red** = **192**, **green** = **192**, **blue** = **255**)







This property represents the color of the signature's background as an RGB value.

In order to not set a color, i.e. keep the rectangle transparent, set the [FillColor](#) to **-1**. This is particularly useful in combination with adding an image to the signature.

Color Examples: Color values are

$\text{color} = \langle \text{red} \rangle + \langle \text{green} \rangle \times 256 + \langle \text{blue} \rangle \times 256 \times 256$,

where $\langle \text{red} \rangle$, $\langle \text{green} \rangle$ and $\langle \text{blue} \rangle$ assume values from 0 to 255.

	Red	255,0,0	255
	Green	0,255,0	65'280
	Blue	0,0,255	16'711'680
	Cyan	0,255,255	16'776'960
	Magenta	255,0,255	16'711'935
	Yellow	255,255,0	65'535
	Black	0,0,0	0
	Grey	128,128,128	8'421'504
	White	255,255,255	16'777'215

6.2.4 FontName1

Property (get, set): String [FontName1](#)
Default: **"Arial"**

This property defines the font used in upper text, i.e. the text that is set by the property [Text1](#). The font can either be specified as a path to the font file, e.g. "C:\Windows\Fonts\arial.ttf", or as a font name, such as "Times New Roman, Bold". When using a font name, the corresponding font must be present in one of the font directories described in chapter [Fonts](#).

6.2.5 FontName2

Property (get, set): String FontName2
Default: FontName1

This property represents the path to the font name used in lower text, i.e. the text that is set by the property [Text2](#). The property works analogously to [FontName1](#).

6.2.6 Font1Mem

Property (set): Variant Font1Mem

Set the font used in upper text (see [FontName1](#)) by passing the font as a memory buffer.

6.2.7 Font2Mem

Property (set): Variant Font2Mem

Set the font used in lower text (see [FontName2](#)) by passing the font as a memory buffer.

6.2.8 FontSize1

Property (get, set): Single FontSize1
Default: 16

Define the font size of the [Text1](#).

6.2.9 FontSize2

Property (get, set): Single FontSize2
Default: 8

Define the font size of the [Text2](#).

6.2.10 ImageFileName

Property (get, set): String ImageFileName

Default: ""

Define the path to an image file that is to be added to the signature. The image is centered and scaled down proportionally to fit into the given rectangle. If the path is **Nothing**, or the image does not exist, the appearance's background is a filled rectangle using the colors [FillColor](#) and [StrokeColor](#).

If you want the appearance to contain the image only and no text, set the property [Text2](#) to a space " ".

6.2.11 Issuer

Property (get, set): String Issuer

Default: ""

Set the issuer of the certificate. The **"Issuer"** corresponds to the common name (CN) of the issuer. In the Windows' certificate store this corresponds to **"Issued by"**.

This property can be used to select the signer certificate for signing (see description of [Cryptographic Provider](#)).

6.2.12 LineWidth

Property (get, set): Single LineWidth

Default: 2

This is the thickness of the line surrounding the visual appearance of the signature.

6.2.13 Location

Property (get, set): String Location

Default: ""

This is the physical location where the signature was added, for example **"Zurich, Switzerland"**.

If this property is set to an empty string, no entry is created.

6.2.14 Name

Property (get, set): String Name

Default: ""

In order to sign a PDF document, a valid, existing certificate name must be provided.

The "Name" corresponds to the common name (CN) of the subject.

In the Windows' certificate store this corresponds to "Issued to".

When using a Windows OS, the certificate must be available in the Windows certificate store. See also chapter [Digital Signatures](#).

This property can be used to select the signer certificate for signing (see description of [Cryptographic Provider](#) in use).

6.2.15 PageNo

Property (get, set): Long PageNo
Default: -1 (last page)

The numbers are counted starting from 1 for the first page.

6.2.16 Provider

Property (get, set): String Provider
Default: (Windows only) "Microsoft Base Cryptographic Provider v1.0"

This property specifies the cryptographic provider used to create and verify signatures.

For more information on the different providers available, see the description in the respective subsection of the section [Cryptographic Provider](#).

- When using the [Windows Cryptographic Provider](#), the value of this property is to be set to a string with the following syntax:

```
"[ProviderType:]Provider[;PIN]"
```

If the name of the provider is omitted, the default provider is used.

Example: "123456" being the pin code:

```
Provider = "Microsoft Base Cryptographic Provider v1.0;123456"
```

```
Provider = ";123456"
```

- When using the [PKCS#11 Provider](#), the value of this property is to be set to a string with the following syntax:

```
"PathToDll;SloId;Pin"
```

Example:

```
Provider = "\\WINDOWS\system32\siacap11.dll;4;123456"
```

- When using any of the service providers, such as the Swisscom All-in signing service, the value of this property is essentially the url of the service endpoint:

```
"http[s]://server.servicedomain.com:8080/url"
```

6.2.17 Reason

Property (get, set): String Reason
Default: ""

Set or get the descriptive text for why the digital signature was added. It is not required in order to create a valid signature.

If this property is set to an empty string, no entry is created.

6.2.18 Rect

Property (get, set): Variant Rect
Default: [0, 0, 0, 0]

Set or get the position and size of the digital signature annotation. The default is an invisible signature.

The position is defined by the four values for the lower-left (x1, y1) and upper-right (x2, y2) corner of the rectangle. The units are PDF points (1 point = 1/72 inch, A4 = 595 x 842 points, Letter = 612 x 792 points) measured from the lower left corner of the page. If either the width or height is zero or negative, an invisible signature is created, i.e. no visible appearance is created for the signature. To create a signature in the lower left corner set the rectangle to [10, 10, 210, 60].

Hint about using this property in programming language that do not support the type **Variant**: In order to find out what type you should use, create a PdfSignature object and look at the default value of the property in the debugger.

6.2.19 SerialNumber

Property (get, set): String SerialNumber

The serial number with the issuer can be used to select a certificate for signing.

This property is a hex string as displayed by the "Serial number" field in the Microsoft Management Console (MMC), e.g. "49 cf 7d d1 6c a9".

This property can be used to select the signer certificate for signing (see description of [Cryptographic Provider](#) in use).

6.2.20 SignerFingerprint

Property (get, set): Variant SignerFingerprint

The sha1 fingerprint of the signer certificate. This property can be used to select the signer certificate for signing (see description of [Cryptographic Provider](#)). After validating a signature, this property contains the validated signature's fingerprint.

6.2.21 SignerFingerprintStr

Property (get, set): String SignerFingerprintStr

The hex string representation of the signer certificate's sha1 fingerprint. This property can be used to select the signer certificate for signing (see description of [Cryptographic Provider](#)).

All characters outside the ranges 0-9, a-f and A-F are ignored. In the Microsoft Management Console, the "Thumbprint" value can be used without conversion, if the "Thumbprint algorithm" is "sha1". E.g. **b5 e4 5c 98 5a 7e 05 ff f4 c6 a3 45 13 48 0b c6 9d e4 5d f5**.

6.2.22 Store

Property (get, set): String Store
Default: **"MY"**

For the [Windows Cryptographic Provider](#) this defines the certificate store from where the signing certificate should be taken. This depends on the OS. The default is **MY**. Other supported values are: **CA** or **ROOT**.

6.2.23 StoreLocation

Property (get, set): Integer StoreLocation
Default: **1**

For the [Windows Cryptographic Provider](#) this defines the location of the certificate store from where the signing certificate should be taken. Supported are:

- 0** Local Machine
- 1** Current User (default)

For more information, see the detailed description of the [Windows Cryptographic Provider](#).

6.2.24 StrokeColor

Property (get, set): Long StrokeColor
Default: **8405056** (red = **64**, green = **64**, blue = **128**)

This is the color of the signature's border line as an RGB value. For examples of RGB color values see the documentation of the [FillColor](#);

In order to not set a color, i.e. keep it transparent, set the `StrokeColor` to `-1`.

6.2.25 SubFilter

Property (get, set): `String SubFilter`

Indicates the encoding of the signature. This value can be set when creating new signatures with [AddSignature](#). The following are common `SubFilter` values:

adbe.pkcs7.detached (PDF 1.6) Legacy PAdES Basic (ETSI TS 102 778, Part 2) signature used for document signatures ([AddSignature](#)).

ETSI.CAdES.detached (PDF 2.0) PAdES signature as specified by European Norm ETSI EN 319 142. This type is used for document signatures ([AddSignature](#)). See chapter [How to Create a PAdES Signature](#) for more information.

6.2.26 Text1

Property (get, set): `String Text1`
Default: `""`

This is the upper text that is added to the signature.

If this property is set to blank, the signature name is added to the upper text line of the visual signature.

In order to position text use the following syntax: `<tab><x>,<y><delimiter><text>`

<code><tab></code>	tabulator
<code><x>,<y></code>	integers
<code><delimiter></code>	Single character such as space
<code><text></code>	Any text string not containing a <code><tab></code>

Example: for Visual Basic .NET

```
Dim sig As New Pdf2PdfAPI.Signature
...
sig.Text1 = Microsoft.VisualBasic.vbTab & "5,50 Peter Pan"
sig.Text2 = Microsoft.VisualBasic.vbTab & "15,25 Signed this document"
```

6.2.27 Text1Color

Property (get, set): `Long Text1Color`
Default: `0` (`black`)

This property defines the color of the upper text, i.e. the text that is set by the property [Text1](#). For examples of RGB color values see the documentation of the [FillColor](#);

6.2.28 Text2

Property (get, set): `String Text2`
Default: `""`

This is the lower text that is added to the signature. The text can be multi-lined by using linefeed ('\n', 0xA).

If this property is set to blank, a text three-line text is constructed that consists of:

- A statement who applied to signature
- The reason of the signature
- The date

See also property [Text1](#). If you want the appearance to not contain any text, set this property to a space " ".

6.2.29 Text2Color

Property (get, set): `Long Text2Color`
Default: `0 (black)`

This property defines the color of the lower text, i.e. the text that is set by the property [Text2](#). For examples of RGB color values see the documentation of the [FillColor](#);

6.2.30 TimeStampCredentials

Property (get, set): `String TimeStampCredentials`
Default: `""`

If a time-stamp server requires authentication, use this property to provide the credentials. Credentials commonly have the syntax `"username:password"`.

6.2.31 TimeStampURL

Property (get, set): `String TimeStampURL`
Default: `""`

The URL of the trusted Time-stamp authority (TSA) from which a Time-stamp shall be acquired. This setting is suggested to be used when applying a Qualified Electronic Signature. Example: `"tsu.my-timeserver.org"`. Applying a Time-stamp requires an online connection to a time server; the firewall must be configured accordingly. In case a web proxy is used, it must be ensured the following MIME types are supported:

`application/timestamp-query`

application/timestamp-reply

If an invalid Time-stamp server address is provided or no connection can be made to the time server, the return code of [Convert](#), [ConvertMem](#), [ConvertStream](#) is false, and the property [ErrorCode](#) is set to [SIG_CREA_E_TSP](#).

6.3 Enumerations

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

6.3.1 TPDFCompliance Enumeration

TPDFCompliance Table

TPDFCompliance	
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

Note that only the values listed above are supported.

6.3.2 TPDFConversionError Enumeration

TPDFConversionError

TPDFConversion Error	Description
ePDFConversionErrorNone	None
ePDFConversionErrorVisualDiff	Visual differences in output file.
ePDFConversionErrorColorants	Resolve name collisions of colorants (PDF/A-2 and PDF/A-3 only).
ePDFConversionErrorOCGRemoved	Remove optional content groups (layers) (PDF/A-1 only).
ePDFConversionErrorTranspRemoved	Remove transparency (PDF/A-1 only).

TPDFConversionError

<code>ePDFConversionErrorEFRemoved</code>	Remove embedded files.
<code>ePDFConversionErrorXMPRemoved</code>	Remove non convertible XMP metadata.
<code>ePDFConversionErrorDocSigned</code>	Conversion of signed document forced removal of signatures. The conversion of a file to PDF/A invalidates all signatures of the input file. For that reason, all signatures are removed and this conversion event is set. Optionally the visual appearances of the signatures can be preserved by setting the property FlattenSignatures to true.
<code>ePDFConversionErrorCorrupt</code>	The input document is corrupt and should be repaired. The errors encountered are printed to the log file. Some errors can be repaired, but it is crucial to review the output file and perform the post analysis.
<code>ePDFConversionErrorFontSubst</code>	Failed to find the same font for embedding, so a similar substitution font was used.
<code>ePDFConversionErrorActionRemoved</code>	Remove interactive elements such as actions or annotations.
<code>ePDFConversionErrorStructureRemoved</code>	Remove logical structure information.

6.3.3 TPDFErrorCode Enumeration

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

The single letter gives an indication of the severity of the error. These are: Success, Error, Warning and Information. In general, an error is returned if an operation could not be completed, e.g. no valid output file was created. A warning is returned if the operation was completed, but problems occurred in the process.

A list of all error codes is available in the C API's header file `bseerror.h`, the javadoc documentation of `com.pdfutils.NativeLibrary.ERRORCODE` and the .NET documentation of `Pdfutils.Pdf.PDFErrorCode`. Note that only a few are relevant for the 3-Heights™ PDF to PDF/A Converter API, most of which are listed here:

TPDFErrorCode Table

TPDFErrorCode	Description
<code>PDF_S_SUCCESS</code>	The operation was completed successfully.
<code>LIC_E_NOTSET</code> , <code>LIC_E_NOTFOUND</code> , ...	Various license management related errors.
<code>PDF_E_FILEOPEN</code>	Failed to open the file.
<code>PDF_E_FILECREATE</code>	Failed to create the file.
<code>PDF_E_PASSWORD</code>	The authentication failed due to a wrong password.

TPDFErrorCode Table

PDF_E_UNKSECHANDLER	The file uses a proprietary security handler, e.g. for a proprietary digital rights management (DRM) system.
PDF_E_XFANEEDSRENDERING	<p>The file contains unrendered XFA form fields, i.e. the file is an XFA and not a PDF file.</p> <p>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.</p>
SIG_CREA_E_SESSION	Cannot create a session (or CSP).
SIG_CREA_E_STORE	Cannot open certificate store.
SIG_CREA_E_CERT	Certificate not found in store.
SIG_CREA_E_INVCERT	The signing certificate is invalid.
SIG_CREA_E_OCSP	Couldn't get response from OCSP server.
SIG_CREA_E_CRL	Couldn't get response from CRL server.
SIG_CREA_E_TSP	Couldn't get response from time-stamp server.
SIG_CREA_E_PRIVKEY	<p>Private key not available.</p> <p>This is usually because a pin is required and was not entered correctly.</p> <p>Also, this error might be returned because there is no private key available for the signing certificate or the key is no properly associated with the certificate.</p> <p>Finally, this error could be the result of choosing a message digest algorithm or signing algorithm which is not supported by the provider.</p> <p>See section Cryptographic Provider for more information.</p>
SIG_CREA_E_SERVER	Server error.
SIG_CREA_E_ALGO	The cryptographic provider does not implement a required algorithm. See section Cryptographic Provider for more information.
SIG_CREA_E_FAILURE	Another failure occurred.

TPDFErrorCode Table

PDF_E_SIGLENGTH	<p>Incorrect signature length.</p> <p>A PDF is signed in a two-step process. First, the output document is created with space reserved for the signature. Second, the actual cryptographic signature is created and written into the space reserved. If the space reserved is too small for the actual signature this error is returned. In general this error should not occur. If it does, the next signing attempt should be successful.</p>
PDF_E_SIGABG	Unable to open signature background image.
PDF_E_CONFORMANCE	The document does not conform to the requested standard (pre analysis).
PDF_E_CONVERSION	Critical conversion errors occurred during conversion. Nonetheless the resulting document is PDF/A compliant. See chapter Conversion Errors for more information on how conversion errors can be handled.
PDF_E_POSTANALYSIS	Output file is not conformant (post analysis). For more information see section Post Analysis .
PDF_E_STOPPED	Cannot convert input file due to compliance problems. The input file is probably corrupt.
PDF_E_LINEARIZATION	Linearization error occurred.
PDF_E_ZUGFERDXML	Failed to add ZUGFeRD invoice file. E.g. because the file is not a valid ZUGFeRD XML invoice or the conformance is not set to PDF/A-3.
PDF_E_INVCOMPLIANCE	Invalid or unsupported PDF conformance. Either the property Compliance has been set to an invalid value or the conformance of the input file is not supported.
PDF_E_OCR	Aborted conversion because of an ocr error.
PDF_E_MISSINGFONT	Unable to convert file to PDF/A because a font that must be embedded is not available in the font directories. See chapter Fonts for more information on resolving this issue.
PDF_E_NOPAGES	Input file contains no pages.
PDF_E_COLLECTION	Unable to convert file to PDF/A-1, because it is a collection (also called PDF Portfolio). Also see property AllowUpgrade .
PDF_E_DOWNGRADE	Input file cannot be converted to meet required conformance level. See property AllowDowngrade .

6.3.4 TPDFInvoiceType Enumeration

Automatic Profile Detection

ePDFInvoiceZugferd ZUGFeRD (version and profile is determined automatically)

ePDFInvoiceFacturX Factur-X (version and profile is determined automatically)

ZUGFeRD 1.0 Profiles

[ePDFInvoiceZugferd1p0Basic](#) ZUGFeRD 1.0 BASIC
[ePDFInvoiceZugferd1p0Comfort](#) ZUGFeRD 1.0 COMFORT
[ePDFInvoiceZugferd1p0Extended](#) ZUGFeRD 1.0 EXTENDED

ZUGFeRD 2.0 Profiles

[ePDFInvoiceZugferd2p0Minimum](#) ZUGFeRD 2.0 MINIMUM
[ePDFInvoiceZugferd2p0BasicWL](#) ZUGFeRD 2.0 BASIC WL
[ePDFInvoiceZugferd2p0Basic](#) ZUGFeRD 2.0 BASIC
[ePDFInvoiceZugferd2p0EN16931](#) ZUGFeRD 2.0 EN 16931 (COMFORT)
[ePDFInvoiceZugferd2p0Extended](#) ZUGFeRD 2.0 EXTENDED

ZUGFeRD 2.1 Profiles

[ePDFInvoiceZugferd2p1Minimum](#) ZUGFeRD 2.1 MINIMUM
[ePDFInvoiceZugferd2p1BasicWL](#) ZUGFeRD 2.1 BASIC WL
[ePDFInvoiceZugferd2p1Basic](#) ZUGFeRD 2.1 BASIC
[ePDFInvoiceZugferd2p1EN16931](#) ZUGFeRD 2.1 EN 16931 (COMFORT)
[ePDFInvoiceZugferd2p1Extended](#) ZUGFeRD 2.1 EXTENDED
[ePDFInvoiceZugferd2p1XRechnung](#) ZUGFeRD 2.1 XRECHNUNG

Factur-X 1.0 Profiles

[ePDFInvoiceFacturX1p0Minimum](#) Factur-X 1.0 MINIMUM
[ePDFInvoiceFacturX1p0BasicWL](#) Factur-X 1.0 BASIC WL
[ePDFInvoiceFacturX1p0Basic](#) Factur-X 1.0 BASIC
[ePDFInvoiceFacturX1p0EN16931](#) Factur-X 1.0 EN 16931 (COMFORT)
[ePDFInvoiceFacturX1p0Extended](#) Factur-X 1.0 EXTENDED

7 Log File

All steps in the diagram from chapter [Process Description](#) can write to the log file. There are three types of messages in the log file: Warnings/Information, Errors and Reports.

7.1 Warnings and Information

Describe the current process step. They do not inhibit the conversion. Prefix: -

Example:

```
- Opening file input.pdf
- Analyzing input.pdf
- Conformance level has been downgraded to level U.
- Performing post analysis for output.pdf.
- Post analysis for output.pdf has been successful.
- File input.pdf converted successfully.
```

7.2 Errors

Inhibit a successful conversion. Prefix: *

Example: The input file cannot be opened

```
* Cannot open file input.jpg.
```

Example: Distinguish critical from non-critical conversion events

Critical conversion events use the prefix * and non-critical - (see chapter [Conversion Errors](#)).

```
- Opening file input.pdf.
- Analyzing input.pdf.
- FontSubst: Substitute font 'Arial-BlackItalic' with multiple master font.
- Embed font 'Verdana'.
- Embed font 'Verdana-Bold'.
* Metadata: xmp:Format :: Value removed because it is not defined in the schema description.
(document metadata)
- Create calibrated color space substitute for DeviceGray. (content of page 1)
- Conversion events:
  * Parts of the XMP metadata could not be repaired and had to be removed.
  - Font substituted.
- Performing post analysis for output.pdf.
- Post analysis for output.pdf has been successful.
* Conversion errors in output.pdf.
```

7.3 Reports

Reports are only created if the corresponding option (Details or Summary) is selected. In detailed reports, each violation is listed with a page number (page 0 = document level), pdf object number, error code, a description, and

a counter of how many times the error occurs. In a summary report, violations that are detected at least once are reported once. Prefix: none.

Example: Details

```
"input.pdf", 0, 53, 0x80410604, "The key Metadata is required but missing.", 1

```

Example: Summary

The document contains fonts without embedded font programs or encoding information (CMAPs).
The document's meta data is either missing or inconsistent or corrupt.
The document doesn't provide appropriate logical structure information.

8 Version History

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

8.1 Changes in Version 6

- Digital Signatures
 - Swisscom All-in Signing Service
 - **New** support for accounts (Identity) based on Swisscom CA 4 Certificate Authorities.
 - **New** support to create PAdES signatures (format ETSI .CAAdES .detached).
 - **Improved** embedding of revocation information (OCSP, CRL, and trust chain) to always use the document security store (DSS)¹⁵.
 - **Changed** the creation of signatures of format ETSI .CAAdES .detached to include revocation information if [EmbedRevocationInfo](#) is **True** and if supported by the cryptographic provider.
 - **Improved** support for new version of the GlobalSign Digital Signing Service. The service endpoint should be updated to `https://emea.api.dss.globalsign.com:8443/v2`.
 - **New** property [NoDSS](#).
- **Improved** repair of corrupt DCT stream data.
- **New** support for ZUGFeRD 2.1.1 electronic invoices (including XRechnung)
- **Improved** creation of annotation appearances.
- **New** support use of fonts that are installed only for the current Windows user.
- **Improved** search algorithm for installed fonts: User fonts under Windows are now also taken into account.
- [Java] **Changed** minimal supported Java language version to 7 [previously 6].
- [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.

Interface Pdf2Pdf

- **New** Property [EmbedAllFonts](#).
- **Changed** behavior of method [AddZUGFeRDXml](#): ZUGFeRD 2.1 is now used instead of the equivalent Factur-X 1.0. ZUGFeRD 2.1 is always preferred over 2.0 and thus profile "EN 16931" is now also supported and not considered ambiguous anymore.
- **Changed** behavior of method [AddInvoiceXml](#): ZUGFeRD 2.1 is now preferred over 2.0 if [ePDFInvoiceZugferd](#) was selected.
- **Changed** behavior of property [SubsetFonts](#). When using **False** embedded fonts are only replaced if they match the installed font.

8.2 Changes in Version 5

- Digital Signatures
 - **New** support to get CRLs using HTTPS and via HTTP redirection.
- **New** support for ZUGFeRD 2.0 hybrid electronic invoices.

¹⁵ Use the property [NoDSS](#) to restore the previous behavior.

- **Improved** conversion of transparent objects to PDF/A-1.
- **Improved** log output for conversion events. Now all conversion events are written to the log file.
- **New** additional supported operating system: Windows Server 2019.
- [PHP] **New** extension PHP 7.3 (non thread safe) for Linux.

Interface Pdf2Pdf

- [.NET, C, Java] **New** method [ConvertStream](#) to convert a stream to PDF/A.
- **New** methods [AddInvoiceXml](#) and [AddInvoiceXmlMem](#) to add XML invoice files with additional options.
- **Deprecated** methods [AddZUGFeRDXml](#) and [AddZUGFeRDXmlMem](#) (replaced by [AddInvoiceXml](#) and [AddInvoiceXmlMem](#)).
- **Changed** behavior of methods [AddZUGFeRDXml](#) and [AddZUGFeRDXmlMem](#): In case of an error, [PDF_E_INVOICEXML](#) is signaled during conversion instead of [PDF_E_ZUGFERDXML](#).

8.3 Changes in Version 4.12

- **Introduced** license feature [Signature](#).
- Digital Signatures
 - **New** support to sign OCSP requests, if required by the OCSP service.
 - **New** support for OCSP requests over HTTPS.
 - **Changed** acceptance criteria for OCSP responses that specify no validity interval (missing nextUpdate field, which is uncommon). Previously a validity interval of 24 hours has been used, now 5 minutes due to Adobe® Acrobat® compatibility.
- **New** support for Factur-X hybrid electronic invoices.
- **New** OCR plugin "abbyy12" for the ABBYY FineReader 12 engine.
- **Changed** behavior when adding additional font directories: The default directories are now always considered.
- **Improved** detection of corrupt embedded fonts, DCT streams and CMap encodings.
- **New** HTTP proxy setting in the GUI license manager.

Interface Pdf2Pdf

- **New** property [ForceEmbeddingOfCMaps](#) to force the embedding of predefined CMaps.
- **Changed** property [TryConvertEmbPDF](#) to only process files whose MIME Type is PDF.
- [PHP] **New** method [convertMem](#).

8.4 Changes in Version 4.11

- **New** font substitution for CJK (Chinese, Japanese, Korean) fonts if an exact match is missing.
- **Improved** conversion of files with optional content (layers) to minimize visual differences while also preserving all content.
- **New** support for the creation of appearance streams for free text annotations that contain rich text content.
- Digital Signatures
 - **New** ability to sign documents that are larger than 2GB (64-bit version only).
- **New** support for reading PDF 2.0 documents.
- **New** support for the creation of output files larger than 10GB (not PDF/A-1).
- **Improved** search in installed font collection to also find fonts by other names than TrueType or PostScript names.
- **Improved** font subsetting of CFF and OpenType fonts.

- **Improved** repair of corrupt image streams.
- **New** treatment of the DocumentID. In contrast to the InstanceID the DocumentID of the output document is inherited from the input document.

Interface Pdf2Pdf

- **New** property `TryConvertEmbPDF` to activate conversion of embedded PDF documents (PDF/A-3 only).
- [PHP] **Removed** the method `Terminate`: It is now called automatically by the “PdfTools” PHP extension and has thereby been rendered obsolete.

8.5 Changes in Version 4.10

- **Improved** conversion of transparent objects to PDF/A-1. E.g. filled paths that are transparent are converted to outlines in order to not cover underlying content when the transparency attribute is removed.
- **Improved** conversion of numbers that are larger than the implementation limit of PDF/A-1.
- **Improved** conversion of logical structure information (PDF/A level A).
- **Changed** behavior: Lock OCGs (layers) that need to be added to user interface (PDF/A-2 and PDF/A-3).
- Digital Signatures
 - **New** support for the new European PAdES norm (ETSI EN 319 142). See chapter “How to Create a PAdES Signature” in the user manual for more information.
 - **New** support for the GlobalSign Digital Signing Service as cryptographic provider to create signatures and time-stamps.
 - **New** signature algorithm RSA with SSA-PSS (PKCS#1v2.1) can be chosen by setting the provider session property `SigAlgo`.
- **New** conversion event `ePDFConversionErrorStructureRemoved` (65536) when logical structure information is removed.
- **New** support for writing PDF objects into object streams. Most objects that are contained in object streams in the input document are now also stored in object streams in the output document. When enabling linearization, however, no objects are stored in object streams.
- **Improved** robustness against corrupt input PDF documents.
- **Improved** annotation appearance generation for polyline, squiggly, and stamp annotations.
- **Removed** the font `ZapfDingbats.ttf` from the product kit as it is not required anymore.
- [C] **Clarified** Error handling of `TPdfStreamDescriptor` functions.

Interface Pdf2Pdf

No functional changes.

Interface PdfSignature

- **New** property `SubFilter` to set signature format, e.g. for new European PAdES norm.

8.6 Changes in Version 4.9

- **New** conversion features, e.g. improved conversion of TrueType font programs, ICC color profiles, or creation of annotation appearances.
- **New** supports for bold font simulation if only non-bold font is available in installed font directories.

- **Changed** behavior: The pre-analysis is now more strict, especially in certain corner cases of the PDF/A ISO Standard for which a conversion is strongly advised.
- **Improved** conversion of .notdef character for PDF/A-2 and PDF/A-3.
- **New** generated conversion event `ePDFConversionErrorVisualDiff` (4):
 - When text in input file is ambiguous.
 - When XFA (XML Forms Architecture) form data is removed.
 - When visual appearance of annotation cannot be created.
- **Improved** support for and robustness against corrupt input PDF documents.
- **Improved** repair of embedded font programs that are corrupt.
- **New** support for OpenType font collections in installed font collection.
- **Improved** metadata generation for standard PDF properties.
- [C] **Changed** return value `pfGetLength` of `TPDFStreamDescriptor` to `pos_t`¹⁶.
- [PHP] **New** Interface for Windows and Linux. Supported versions are PHP 5.6 & 7.0 (Non Thread Safe). The Pdf2PdfAPI 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 4.8

- **New** conversion features, e.g. improved conversion of corrupt data such as fonts, text, or form XObjects.
- **New** conversion event `ePDFConversionErrorActionRemoved` (32768) when interactive elements such as actions or annotations are removed.
- **New** bold font simulation used when substituting bold with non-bold font.
- **Improved** creation of annotation appearances to use less memory and processing time.
- **Added** repair functionality for TrueType font programs whose glyphs are not ordered correctly.

Interface Pdf2Pdf

- [.NET, C, COM, Java] **Changed** method `ConvertMem` which now also returns the output file in case of `PDF_E_CONVERSION` and `PDF_E_POSTANALYSIS`.
- [.NET, C, COM, Java] **New** property `ProductVersion` to identify the product version.
- [.NET] **Deprecated** method `GetLicenseIsValid`.
- [.NET] **New** property `LicenseIsValid`.

¹⁶ This has no effect on neither the .NET, Java, nor COM API

9 Licensing, Copyright, and Contact

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

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

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

Contact

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