User Manual

3-Heights®
PDF Security API

PDF/A conform

Version 6.27.10

pdftools

Contents

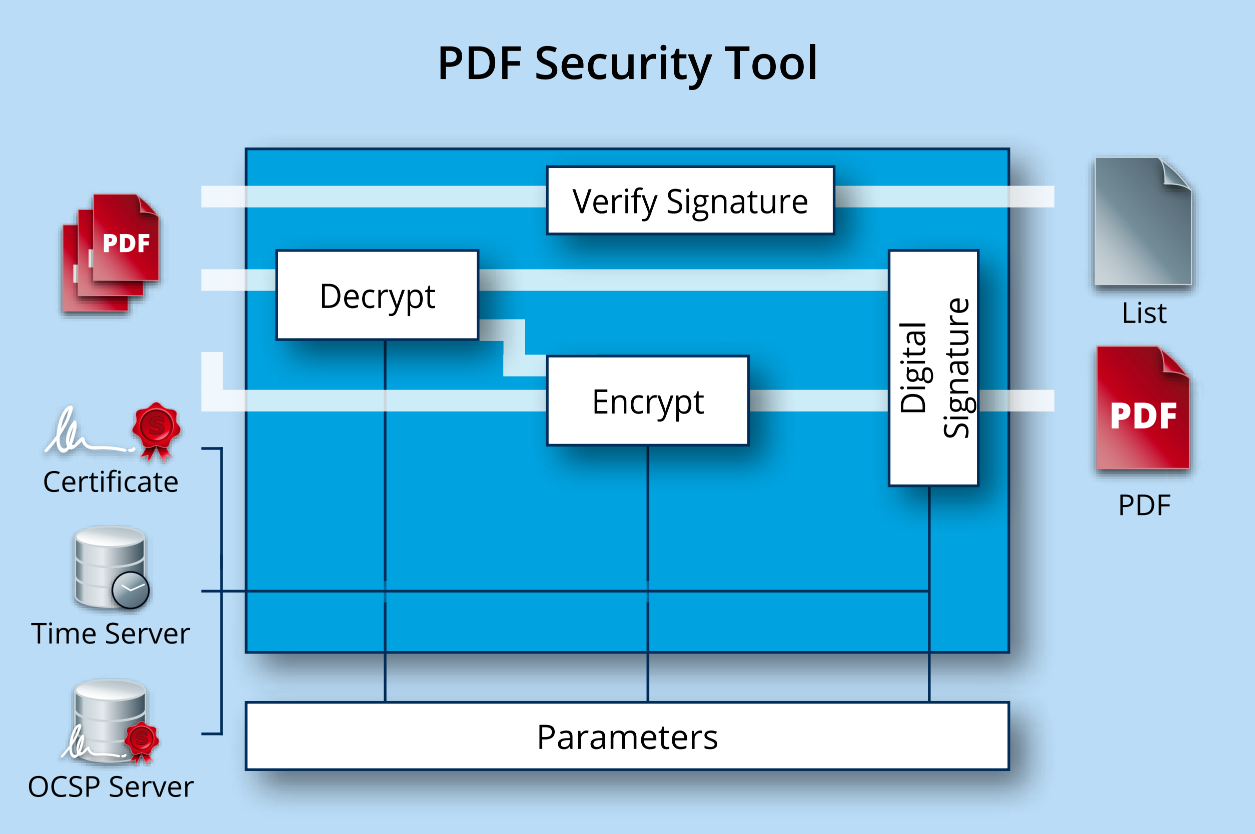
1 Introduction

1.1 Description

The 3-Heights® PDF Security API enables the application of digital signatures to PDF documents and their subsequent protection through setting passwords and user authorizations.

Both standard signatures and qualified signatures that use signature cards (“smartcards”, “USB tokens”, “HSM”) can be used.

PDF documents used in professional circumstances contain important information that needs to be protected against misuse and unintentional alteration. This is achieved by protecting PDF documents through encryption and user authorization rights.



When exchanging electronic documents, the ability to ascertain that a document is authentic and has not been manipulated on its way from sender to recipient is of particular importance. This is only achievable through the use of electronic signatures.

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

1.2 Functions

The 3-Heights® PDF Security API enables users to encrypt and—if the passwords are known—decrypt PDF documents. The tool can set and cancel all known PDF user authorizations. For instance, it can set an owner password so that only authorized users can edit and change the document. A user password ensures that only authorized users have access to the document’s content. The tool’s signature module allows the user to apply, read, and verify both classic digital signatures and MDP (modification detection and prevention) signatures. The visibility and visual appearance of digital signatures can be adapted to suit requirements. The tool also supports customized signature handlers and types.

1.2.1 Features

- Apply simple, advanced, and qualified electronic signatures
  - PDF/A conforming signatures
  - Support European signature standards
  - Signature types
    - Document signatures to “digitally sign” documents
    - Modification detection & prevention (MDP) signatures to “certify” documents
    - Document timestamp signatures to “timestamp” documents
  - Apply PAdES-B-LTA (long-term availability and integrity of validation material) and PAdES-LTV (Long-Term Validation) signatures
    - Embedded trust chain, timestamp, and revocation information (OCSP, CRL)
    - Extend the longevity of existing signatures
    - Add signature validation material to the document security store (DSS)
  - Add an optional visual appearance of the signature (page, size, color, position, text, background image, etc.)
  - Cache OCSP, CRL, and other data for mass-signing
  - Various types of cryptographic providers
    - Windows certificate store
    - Hardware such as hardware security module (HSM), smartcards, and USB tokens
    - Online signature services
      - myBica Digital Signing Service
      - Swisscom All-in Signing Service
      - GlobalSign Digital Signing Service
      - QuoVadis sealsign
    - Custom signature handler plugin interface
  - Mass-signing of documents
  - Multiple signatures
- Extract digital signatures
  - Validate digital signatures
  - Remove digital signatures
  - Extract signed version (revision) of document
- Encrypt and decrypt PDF documents
  - Set document restrictions, including:
    - Print document
    - Modify document content
    - Extract or copy content
    - Add comments
    - Fill in form fields
    - Extract content for accessibility
    - Assemble documents
    - Print in high resolution
  - Set crypt and stream filters
  - Set encryption strength
  - Set owner and user password
- Stamping
  - Stamp text, images, or vector graphics
  - Add hyperlinks

  - Add PDF/A conforming stamps
  - Modify existing stamps
  - Stamping of signed documents preserves existing signatures
  - Stamp on layer
- Set document metadata
- Optimize for the web (linearize) (not for PDF 2.0)
- Read input from and write output document to file, memory, or stream

1.2.2 Formats

Input formats

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

Output formats

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

1.2.3 Conformance

Standards:

- ISO 32000-1 (PDF 1.7)
- ISO 32000-2 (PDF 2.0)
- ISO 19005-1 (PDF/A-1)
- ISO 19005-2 (PDF/A-2)
- ISO 19005-3 (PDF/A-3)
- PAdES (ETSI EN 319 142) signature levels B-B, B-T, B-LT, B-LTA, CMS
- Legacy PAdES baseline signature (ETSI TS 103 172) B-Level, T-Level, LT-Level, and LTA-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)

1.3 Interfaces

The following interfaces are available:

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

[1] Limited supported OS versions. Operating systems

1.4 Operating systems

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

- Windows Client 7+ | x86 and x64
- Windows Server 2008, 2008 R2, 2012, 2012 R2, 2016, 2019, 2022 | x86 and x64
- Linux:
  - Red Hat, CentOS, Oracle Linux 7+ | x64
  - Fedora 29+ | x64
  - Debian 8+ | x64
  - Other: Linux kernel 2.6+, GCC toolset 4.8+ | x64
- macOS 10.10+ | x64

'+' indicates the minimum supported version.

1.5 How to best read this manual

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

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

1.6 Digital signatures

1.6.1 Overview

Digital signature is a large and slightly complex topic. This chapter gives an introduction to digital signatures and describes how the 3-Heights® PDF Security 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
QES	Qualified electronic signature
TSA	Timestamp Authority
TSP	Timestamp 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 (timestamp)

Digitally signing a document requires a certificate and its private key. How to access and use a certificate is described in 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 explanation). Information such as name of the signer, reason, date, and 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 Security 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 fails and reports 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 Security API supports the following types of digital signatures:

Document signature A document signature type digital signature checks the integrity of the signed part of the document and authenticates 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 signature A modification detection and prevention signature detects 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 timestamp signature A timestamp signature provides evidence that the document existed at a specific time and protects the document's integrity. One or more document timestamp 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 (AdES) - "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 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 can create a self-signed certificate issued by "Peter Pan" and issued to "Peter Pan". Using this certificate, a person can 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. 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) for the country, such as VeriSign, SwissSign, QuoVadis. 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 timestamp (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 timestamp).
- Legal processes that require a QES are supported.

> **Note:** A timestamp can be added to any type of signature. OCSP/CRL responses are also available for some advanced certificates.

1.6.5 Creating electronic signatures

This is a simple example of how to create an electronic document signature. More detailed examples and examples for other types of electronic signatures can be found in Creating 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 timestamp required to prove that the signature itself existed at a certain date and time?
    - Should validation information be embedded 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, it is recommended you 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. Set up 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, as 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: Timestamp service URL (e.g. "http://server.mydomain.com:80/tsa")
4. Optional: Timestamp 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 Security API applies PDF/A conforming signatures. This means if a PDF/A document is digitally signed, it retains PDF/A conformance.

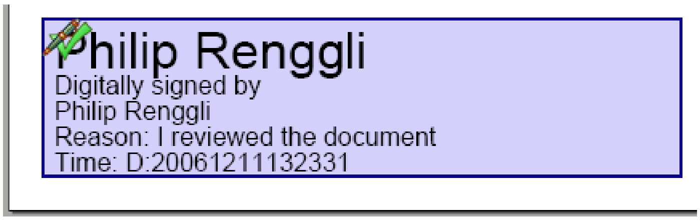
To add an electronic document signature with the 3-Heights® PDF Security API, the following steps need to be done:

1. Create a new `Signature` object
2. Provide the name of the certificate to be used, as value of the `Signature`'s name. The name of the certificate corresponds to the value "Issued to:".
3. If the certificate's private key is PIN protected, pass the PIN in the provider configuration.
4. Set additional parameters, such as the reason why the signature is applied, etc.

In C#, the four steps above look like this:

```
using (Secure doc = new Secure())
{
    if (!doc.Open("input.pdf", ""))
        throw new Exception("Document cannot be opened: " + doc.ErrorMessage);

    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

        doc.AddSignature(signature);
    }

    if (!doc.SaveAs("output.pdf", "", "", PDFPermission.ePermNoEncryption, 0, "", ""))
        throw new Exception("Unable to sign document: " + doc.ErrorMessage);

    doc.Close();
}
```

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 Security API comes as a ZIP archive or as a NuGet package.

To install the software, proceed as follows:

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

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

4. The usage of the NuGet package is described in section NuGet package.
5. (Optional) Register your license key using the License management.
6. Identify the interface you are using. Perform the specific installation steps for that interface described in Interface-specific installation steps.
7. Ensure the cache directory exists as described in Special directories.
8. If you want to sign documents, set up your cryptographic provider as described in Cryptographic provider.
9. If you want to stamp text, set up the fonts required as described in Fonts.

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 Security API provides two interfaces:

- Java interface
- Native C interface

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

File description

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

   If the previous step reports any missing libraries, you have two options:

   a. Download an archive that is linked to a different version of the GNU shared libraries and verify whether they are available on your system. Use any version whose requirements are met. Note that this option is not available for all platforms.

   b. Use your system's package manager to install the missing libraries. It usually suffices to install the package `libstdc++6`.
3. Create a link to the shared library from one of the standard library directories, e.g.

   ```
   ln -s /opt/pdf-tools.com/bin/x64/libPdfSecureAPI.so /usr/lib
   ```
4. Optionally, register your license key using the license manager.
5. Identify the interface you are using. Perform the specific installation steps for that interface described in Interface-specific installation steps.
6. Ensure the cache directory exists as described in Special directories.
7. If you want to sign documents, set up your cryptographic provider as described in Cryptographic provider.
8. If you want to stamp text, set up the fonts required as described in Fonts.

2.2.2 macOS

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

```
ln libPdfSecureAPI.dylib libPdfSecureAPI.jnilib
```

2.3 ZIP archive

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

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

2.3.1 Development

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

Req. The file is required for this interface.

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

Doc. The file is for documentation only.

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

Files for development

Name	.NET	Java	COM	C
bin\‹platform›\PdfSecureAPI.dll	Req.	Req.	Req.	Req.
bin*NET.dll	Req.			
bin*NET.xml	Doc.			

Files for development

Name	.NET	Java	COM	C
doc*.pdf	Doc.	Doc.	Doc.	Doc.
doc\PdfSecureAPI.idl			Doc.	
doc\javadoc\*.*		Doc.		
include\pdfsecureapi_c.h				Req.
include*.*				Opt.
jar\SECA.jar		Req.		
lib\‹platform›\PdfSecureAPI.lib				Req.[2]
samples*.*	Doc.	Doc.	Doc.	Doc.

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

File description

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

2.3.2 Deployment

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

[2] Not required for Linux or macOS.

[3] These files must reside in the same directory as PdfSecureAPI.dll.

Files for deployment

Name	.NET	Java	COM	C
`bin\‹platform›\PdfSecureAPI.dll`	Req.	Req.	Req.	Req.
`bin\*NET.dll`	Req.			
`jar\SECA.jar`		Req.		

The deployment of an application works as described below:

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

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

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

```
copy securer.exe %targetlocation%\.
copy PdfSecureAPI.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%\PdfSecureAPI.dll.
```

2.4 NuGet package

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

Installation

The package `PdfTools.PdfSecure 6.27.10` is available on nuget.org. Right-click on your .NET project in Visual Studio and select “Manage NuGet Packages...”. Finally, select the package source “nuget.org” and navigate to the package `PdfTools.PdfSecure 6.27.10`.

Development

The package `PdfTools.PdfSecure 6.27.10` contains .NET libraries with versions .NET Standard 1.1, .NET Standard 2.0, and .NET Framework 2.0, and native libraries for Windows, macOS, and Linux.

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

To use the software, you must first install a license key for the 3-Heights® PDF Security 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 Security API component in your COM application program, you have to register the component using the `regsvr32.exe` program that is provided with the Windows operating system. The following command shows how to register the PdfSecureAPI.dll. In Windows Vista and later, the command needs to be executed from an administrator shell.

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

2.5.2 Java interface

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

For compilation and execution

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

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

For execution

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

Additionally, the library PdfSecureAPI.dll needs be in one of the system's library directories[5] or added to the Java system property java.library.path. You can add the library by either adding it dynamically at program startup before using the API, or by specifying it using the switch -Djava.library.path when starting the Java VM. Choose the correct subdirectory (x64 or Win32 on Windows) depending on the platform of the Java VM[6].

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

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

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

2.5.3 .NET interface

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

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

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

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

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

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

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

2.5.4 C interface

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

2.6 Uninstall, Install a new version

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

[5] On Windows defined by the environment variable PATH, and on Linux defined by LD_LIBRARY_PATH.

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

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

2.7 Note about the evaluation license

With the evaluation license, the 3-Heights® PDF Security 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 is deleted after termination of the program.

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

Windows

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

Linux and macOS

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

2.8.2 Cache directory

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

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

Windows

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

Linux and macOS

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

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

2.8.3 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 always take precedence over system fonts.

Windows

1. %SystemRoot%\Fonts
2. User fonts listed in the registry key \HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Fonts. This includes user specific fonts from C:\Users\<user>\AppData\Local\Microsoft\Windows\Fonts and app specific fonts from C:\Program Files\WindowsApps
3. Fonts directory, which must be a direct subdirectory of where PdfSecureAPI.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 Security API requires a valid license in order to run correctly. If no license key is set or the license is not valid, then most of the interface elements documented in Interface reference fail with an error code and error message indicating the reason.

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

3.1 License features

The functionality of the 3-Heights® PDF Security API contains two areas to which the following license features are assigned:

`Signature` Create, validate, and enhance signatures.

`Stamping` Apply and modify stamps.

A license can include an arbitrary set of these features. The presence of any 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 which license features.

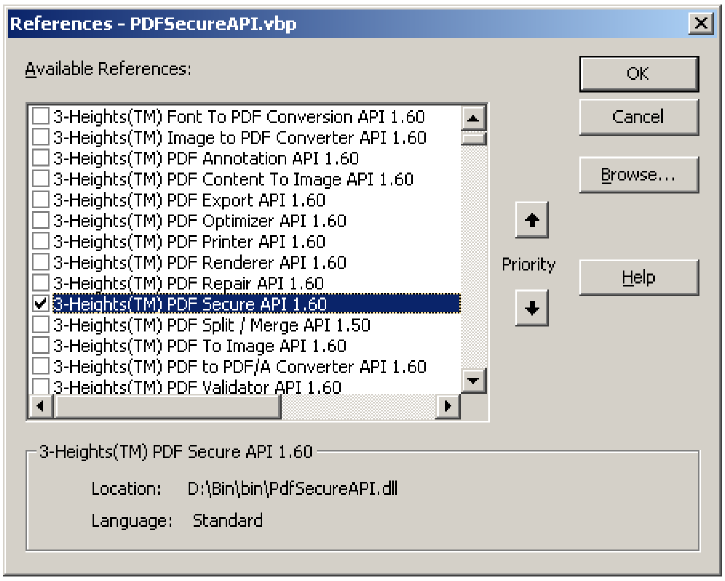
4 Programming interfaces

4.1 Visual Basic 6

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

If you start from scratch, perform these steps:

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



References - PDFSecureAPI.vbp

Available References:

- ☐ 3-Heights(TM) Font To PDF Conversion API 1.60
- ☐ 3-Heights(TM) Image to PDF Converter API 1.60
- ☐ 3-Heights(TM) PDF Annotation API 1.60
- ☐ 3-Heights(TM) PDF Content To Image API 1.60
- ☐ 3-Heights(TM) PDF Export API 1.60
- ☐ 3-Heights(TM) PDF Optimizer API 1.60
- ☐ 3-Heights(TM) PDF Printer API 1.60
- ☐ 3-Heights(TM) PDF Renderer API 1.60
- ☐ 3-Heights(TM) PDF Repair API 1.60
- ☑ 3-Heights(TM) PDF Secure API 1.60
- ☐ 3-Heights(TM) PDF Split / Merge API 1.50
- ☐ 3-Heights(TM) PDF To Image API 1.60
- ☐ 3-Heights(TM) PDF to PDF/A Converter API 1.60
- ☐ 3-Heights(TM) PDF Validator API 1.60

3-Heights(TM) PDF Secure API 1.60

Location: D:\Bin\bin\PdfSecureAPI.dll

Language: Standard

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 Secure As New PDFSECUREAPILib.PdfSecure
   Secure.Open "C:\input.pdf", ""
   Secure.SaveAs "C:\output.pdf", "", "pwd", ePermPrint, 40
   Secure.Close
End Sub
```

And that's all—four lines of code. Create the object, open the input file, create the output file with no user password, owner password "owner", allow printing, and use 40 bit encryption key.

Example: More advanced

The following Visual Basic 6 sample assumes an interface with:

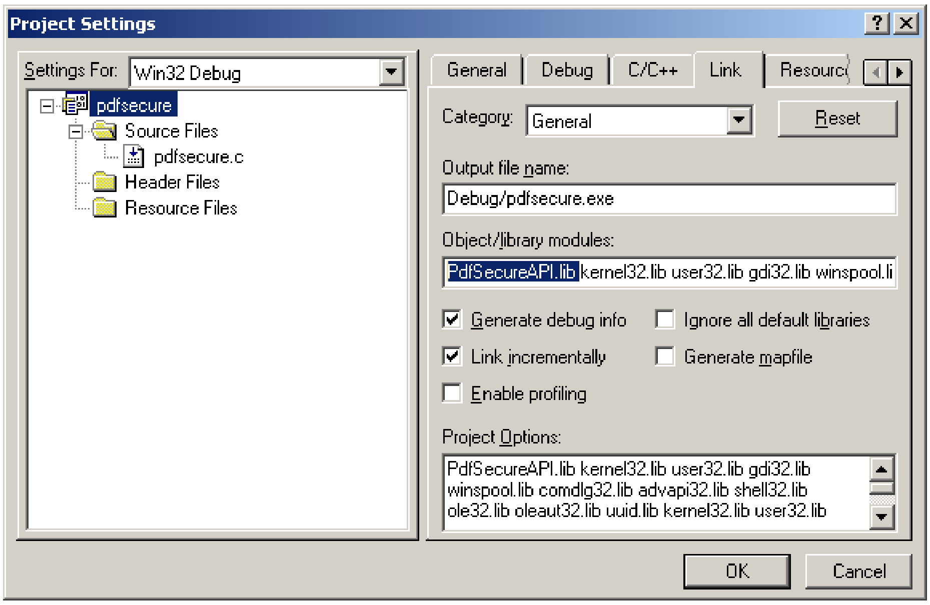
- Text fields (txt*) for the input and output file names, as well as the passwords.
- Check boxes (chk*) with a value to be set to 0 or 1 for all the permission flags.

```
Private Sub CreateOutput_Click()
   Dim doc As New PDFSECUREAPILib.PdfSecure
   Dim iPerm As Integer
   done = doc.Open(txtInput.Text, txtPwd.Text)
   ' Open the input fil
   If Not done Then
      If doc.ErrorCode = PDF_E_PASSWORD Then
         MsgBox "Input file is encrypted and Password not correct."
      Else
         MsgBox "Couldn't open input file."
      End If
      Exit Sub
   End If
   ' Set the permissions
   iPerm = chkPrint.Value * ePermPrint _
       + chkModify.Value * ePermModify _
       + chkCopy.Value * ePermCopy _
       + chkAnnot.Value * ePermAnnotate _
       + chkFill.Value * ePermFillForms _
       + chkExtr.Value * ePermSupportDisabilities _
       + chkAssemble.Value * ePermAssemble _
       + chkDPrint * ePermDigitalPrint
   iKey = 128
   ' Save the output file
   If Not doc.SaveAs(txtOutput.Text, txtUser.Text, txtOwner.Text, iPerm, iKey)
      Then
         MsgBox "Output file could not be created."
   End If
   done = doc.Close
End Sub
```

4.2 C/C++

To use the 3-Heights® PDF Security API in a C project, the following steps should be done. (Note: Steps and screenshots are specifically described for the MS Studio 6)

1. Add the header files `pdfsecureapi_c.h` and `pdfsecuritydecl.h` to the include directories.
2. Link to the object file library. (Windows: `PdfSecureAPI.lib`)



Project Settings

Settings For: Win32 Debug

- pdfsecure
  - Source Files
    - pdfsecure.c
  - Header Files
  - Resource Files

Category: General

Output file name: Debug/pdfsecure.exe

Object/library modules: PdfSecureAPI.lib kernel32.lib user32.lib gdi32.lib winspool.li

☑ Generate debug info ☐ Ignore all default libraries

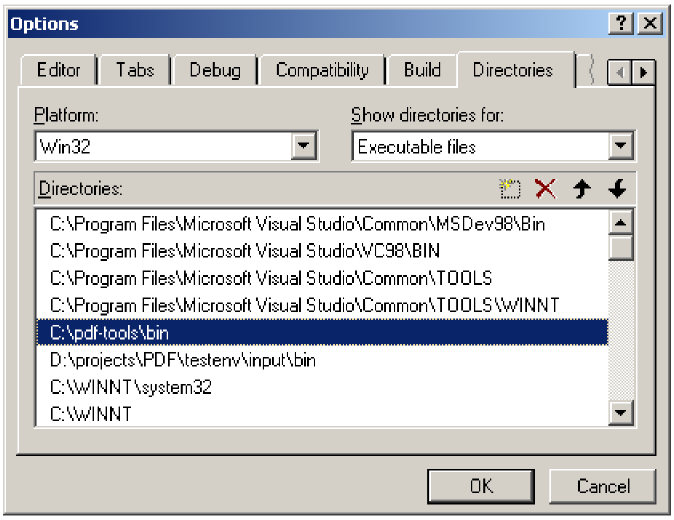
☑ Link incrementally ☐ Generate mapfile

☐ Enable profiling

Project Options:

PdfSecureAPI.lib kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib kernel32.lib user32.lib

3. Add the path where the dynamic link library PdfSecureAPI.dll resides to the “Executable files directories”. For example, as shown in the screenshot below. In most cases, it suffices to simply add it to the environment variable Path.



Platform: Win32

Show directories for: Executable files

Directories:

```
C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin
C:\Program Files\Microsoft Visual Studio\VC98\BIN
C:\Program Files\Microsoft Visual Studio\Common\TOOLS
C:\Program Files\Microsoft Visual Studio\Common\TOOLS\WINNT
C:\pdf-tools\bin
D:\projects\PDF\testenv\input\bin
C:\WINNT\system32
C:\WINNT
```

There is a C sample available within the ZIP archive of the evaluation and release version that shows how to decrypt and encrypt a PDF document, as well as how to add a digital signature. The C sample below is much simpler and does not add a digital signature.

Before the C interface can be used to create objects, it must be initialized once. This is done using PdfSecureInitialize, to un-initialize use PdfSecureUnInitialize. Other than that, equal call sequences as in the COM interface can be used.

```c
#include <stdio.h>
#include "pdfsecureapi_c.h"
#include "pdfsecuritydecl.h"
int main(int argc, char* argv[])
{
    TPdfSecure* pPdfSecure;
    PdfSecureInitialize();

    pPdfSecure = PdfSecureCreateObject();
    PdfSecureOpen(pPdfSecure, argv[1], "");
    PdfSecureSaveAsA(pPdfSecure, argv[2], "", "pwd", ePermPrint, 128, "", "");
    PdfSecureClose(pPdfSecure);

    PdfSecureDestroyObject(pPdfSecure);
    PdfSecureUnInitialize();
    return 0;
}
```

4.3 .NET

There should be at least one .NET sample for MS Visual Studio available in the ZIP archive of the Windows version of the 3-Heights® PDF Security API. The easiest to quickly start is to refer to this sample.

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

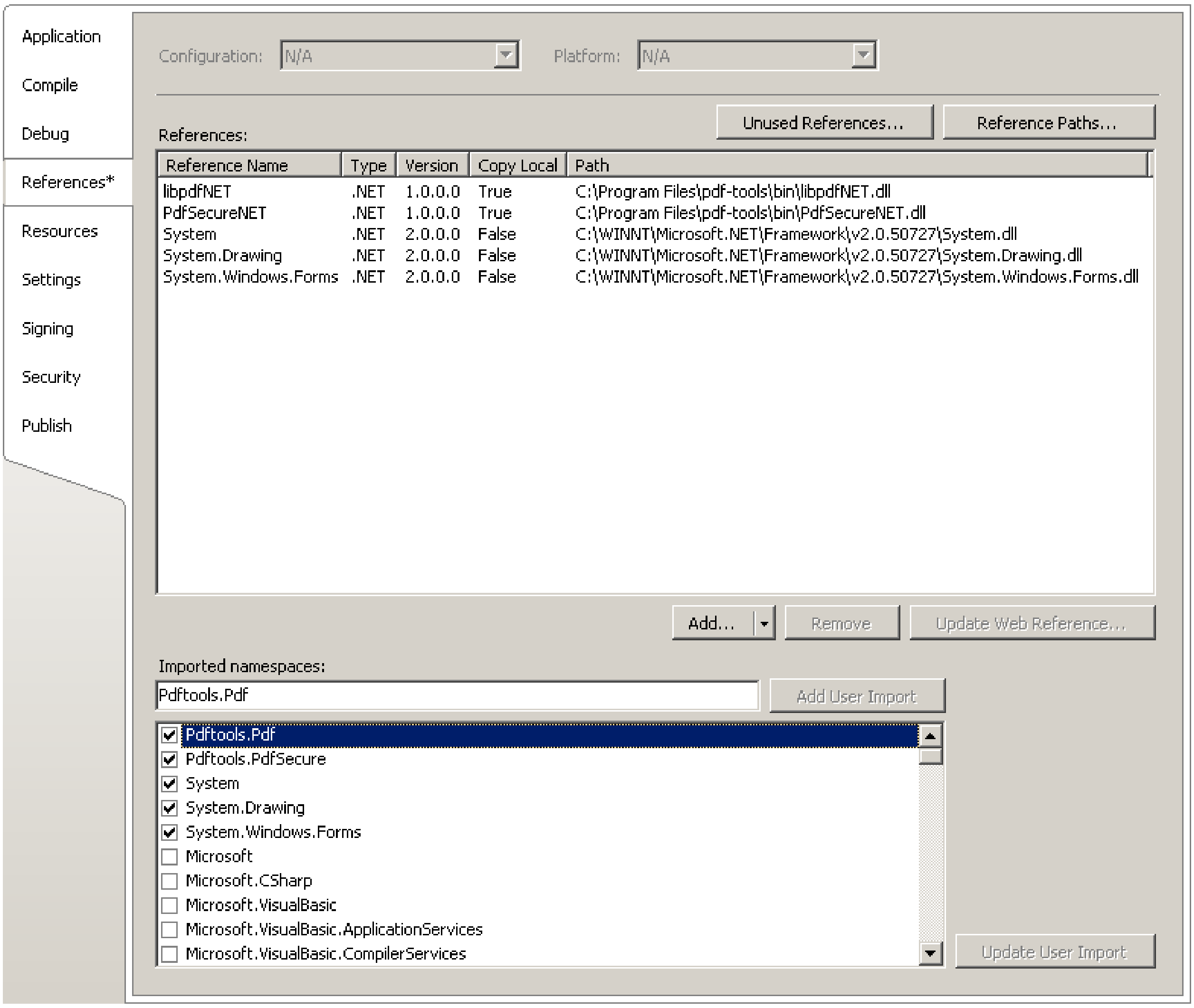
1. Start Visual Studio and create a new C# or VB project.
2. Add references to the NuGet package PdfTools.PdfSecure 6.27.10, as described in NuGet package.

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

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

4.3.1 Visual Basic

3. Double-click “My Project” to view its properties. On the left hand side, select the menu “References”. The .NET assemblies you added before should show up in the upper window. In the lower window, import the namespaces `Pdftools.Pdf`, and `Pdftools.PdfSecure`.
   You should now have settings similar as in the screenshot below:



References:

Reference Name	Type	Version	Copy Local	Path
libpdfNET	.NET	1.0.0.0	True	C:\Program Files\pdf-tools\bin\libpdfNET.dll
PdfSecureNET	.NET	1.0.0.0	True	C:\Program Files\pdf-tools\bin\PdfSecureNET.dll
System	.NET	2.0.0.0	False	C:\WINNT\Microsoft.NET\Framework\v2.0.50727\System.dll
System.Drawing	.NET	2.0.0.0	False	C:\WINNT\Microsoft.NET\Framework\v2.0.50727\System.Drawing.dll
System.Windows.Forms	.NET	2.0.0.0	False	C:\WINNT\Microsoft.NET\Framework\v2.0.50727\System.Windows.Forms.dll

Imported namespaces: Pdftools.Pdf

☑ Pdftools.Pdf
☑ Pdftools.PdfSecure
☑ System
☑ System.Drawing
☑ System.Windows.Forms
☐ Microsoft
☐ Microsoft.CSharp
☐ Microsoft.VisualBasic
☐ Microsoft.VisualBasic.ApplicationServices
☐ Microsoft.VisualBasic.CompilerServices

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

Example:

```
Dim doc As New PdfSecure.Secure
Dim sig As New PdfSecure.Signature
doc.Open(...)
...
If Not doc.SaveAs("C:\temp\output.pdf", _
         "", _
```

```
        "pwd", _
        PDFPermission.ePermPrint, _
        128, _
        "V2", _
        "V2") = True Then
```

4.3.2 C#

3. Add the following namespaces:

Example:

```
using Pdftools.Pdf;
using Pdftools.PdfSecure;
```

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

Example:

```
using (Secure doc = new Secure())
{
    doc.Open(...)
    using (Signature sig = new Signature())
    {
        ...
        doc.AddSignature(sig)
        ...
    }
}
```

4.3.3 Deployment

This is a guideline on how to distribute a .NET project that uses the 3-Heights® PDF Security 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 Security API's .NET assemblies (`*NET.dll`), as well as the native library (`PdfSecureAPI.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 `PdfSecureAPI.dll` is found at execution time, and that the native library's format (32 bit versus 64 bit) matches the operating system.
4. The output directory may contain multiple versions of the native library, e.g. for Windows 32 bit, Windows 64 bit, MacOS 64 bit, and Linux 64 bit. Only the versions that match the target computer's operating system need be deployed.
5. If required by the application, optional DLLs must be copied to the same folder. See Deployment for a list and description of optional DLLs.

4.3.4 Troubleshooting: TypeInitializationException

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

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

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

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

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

Troubleshooting: DllNotFoundException

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

Resolve this by performing one of these actions:

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

Troubleshooting: BadImageFormatException

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

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

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

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

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

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

5 User guide

5.1 Overview of the API

5.1.1 About the 3-Heights® PDF Security API

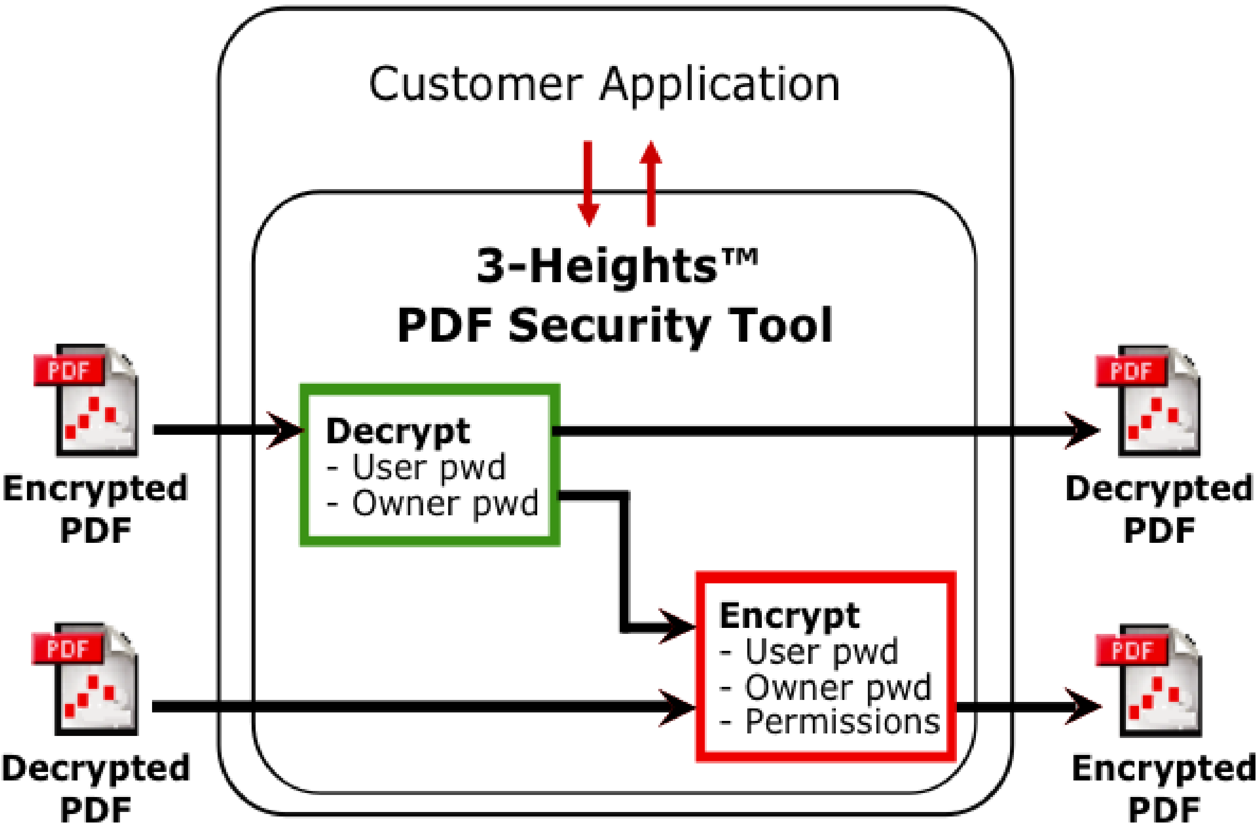
The 3-Heights® PDF Security API provides three key functionalities related to security in PDF documents:

1. Deal with encryption, decryption, and access permissions of PDF documents
2. Deal with digital signatures
3. Apply stamps to PDF documents

These three functionalities can be combined; they however are not closely related. Encryption and digital signature are discussed in Encryption and Digital signatures, respectively.

5.2 About the API

The 3-Heights® PDF Security API requires a PDF document as input. In this manual, that document is referred to as input document. In the graphic below that's the document on the left hand side. The document can be opened from file or from memory. If the document is encrypted, it is in a first step decrypted.



In the next step, application specific operations are applied. These can be setting new passwords and access permissions or add a digital signature (not shown in graphic).

After that, a new PDF document is created according to the defined settings. In this manual, the new resulting document is referred to as output document. The input document is never changed by the 3-Heights® PDF Security API. Thus, the output document must be a new document. It is not possible to directly overwrite the input document.

5.3 Encryption

5.3.1 Encryption and how it works in PDF

A PDF document can be encrypted to protect its contents from unauthorized access. The encryption process applies encryption to all streams (e.g. images) and strings, but not to other items in the PDF document. This means the structure of the PDF document is accessible, but the content of its pages is encrypted.

When encryption is used in PDF, a security handler must be selected. The 3-Heights® PDF Security API always uses the standard security handler that, according to the PDF Specification, has to be supported by any software that can process encrypted PDF documents.

For more detailed information about PDF encryption in general, see PDF Reference, chapter 3.5.

5.3.2 Owner password and user password

The standard security handler allows access permissions and up to two passwords to be specified for a document: An owner password and a user password.

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

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

The following table shows the four possible combinations of passwords and how an application processing such a PDF document behaves.

Owner and user passwords

UserPwd	OwnerPwd	Behavior
none	none	Everyone can read. Everyone can change security settings. (No encryption)
none	set	Everyone can read. The user password is an empty string. Owner password required to change security settings.
set	none	User password required to read. The owner password is equal to the user password. User password required to change security settings.
set	set	User or owner password required to read. Owner password required to change security settings.

5.3.3 Permission flags

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

These access permission flags are:

- Modifying the content of the document
- Copying or extracting text and graphics from the document
- Adding or modifying text annotations and interactive form fields
- Printing the document (low or high quality)
- Filling in forms and digitally signing the document
- Assembling the document (inserting, rotating, deleting pages, etc.)

5.3.4 Encrypting a PDF document

If either of the passwords or permission flags is set, the document is encrypted.

If only a user password is set, but no owner password and no permission flags, the owner password is equal to the user password and all permissions are granted.

In the 3-Heights® PDF Security API, the passwords and permission flags are provided as parameters of the `SaveAs` function. The PDF Specification accepts an empty string as password. PDF applications by default try to open documents with the empty string password.

To encrypt a document and protect it against any manipulations other than printing, the document must have an owner password and the print permission flag set. In Visual Basic, a SaveAs call would look like this:

```
SaveAs("C:\temp\output.pdf", "", "ownerpwd", ePermPrint)
```

To encrypt a document similar as above, but in addition also have the application prompt the user for a password to open and read the document, you can add a user password as additional parameter in the SaveAs function:

```
SaveAs("C:\temp\output.pdf", "userpwd", "ownerpwd", ePermPrint)
```

To not encrypt a document at all, set empty passwords and `ePermNoEncryption` (`-1`) for permission flags:

```
SaveAs("C:\temp\output.pdf", "", "", ePermNoEncryption)
```

5.3.5 Reading an encrypted PDF document

A PDF document that is not encrypted or protected with an owner password only can be read and decrypted by the 3-Heights® PDF Security API's `Open` function without providing a password.

In Visual Basic, it looks like this:

```
Open("C:\temp\input.pdf", "")
```

A PDF document that is protected by a user password can only be opened if either the user or the owner password is provided as parameter in the `Open` function. Technically, it does not matter later on which of the two passwords was provided, because both grant full access to the document. However, it is up to the application programmer to distinguish between input documents that are password protected or not.

5.3.6 How secure is PDF encryption?

Any PDF application that is to process or display a PDF document must be able to read and decrypt the contents of the pages to be able to display them. Technically, it cannot display an encrypted text or image without first decrypting it. A PDF application program has therefore full access to any PDF document it can decrypt and display.

PDF application programs such as all products of the PDF Security API family or Adobe Acrobat, can open and decrypt PDF documents that have an owner password but no user password, without knowing that password. Otherwise, they couldn't display the document. The application at that point has full access to the document. However, this does not imply the user of this application is given the same access rights. The user should only be given the access permissions defined by the permission flags and the password provided. Any PDF application that behaves different from that can allow for changing the security settings or completely removing encryption from the document as long as the original document does not have a user password.

The user password protects the document, so that it only can be opened if the user or owner password is known. No PDF application program can open a user-password protected PDF document without providing the password. The security of such a document, however, strongly depends on the password itself. Like in most password-related situations, insecure passwords can easily be found programmatically. For example, a brute force attempt testing all passwords that either exist as word in a dictionary or have less than six characters only takes minutes.

5.4 Fonts

Some features of the 3-Heights® PDF Security API require fonts to be installed, e.g. for stamping text or the creation of the visual appearance of digital signatures.

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 Security API is run under that user and the user profile is loaded.

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 Security 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.

5.5 Cryptographic provider

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

The 3-Heights® PDF Security API can use various different cryptographic providers. The following list shows the provider that can be used for each type of signing certificate.

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.

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, see 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 stores, see 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 Security API can use various different services. The configuration is explained in the following sections of this documentation:

- myBica Digital Signing Service
- Swisscom All-in Signing Service
- GlobalSign Digital Signing Service
- QuoVadis sealsign

Custom signature handler If you want to create the cryptographic part of the signature yourself, i.e. you want to implement the cryptographic provider yourself, you can register a Custom signature handler. This is described in the respective subsection.

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›` 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 `siecap11.dll`
- The IBM 4758 cryptographic coprocessor uses `cryptoki.dll`
- Devices from Aladdin Ltd., use `etpkcs11.dll`
- For SafeNet Luna, HSM use `cryptoki.dll` an Windows or `libCryptoki2_64.so` on Linux/UNIX.
- For Securosys SA, Primus HSM or CloudsHSM, use `primusP11.dll`[7] on Windows and `libprimusP11.so`[7] on Linux.
- For Google Cloud HSM (Cloud KMS), use `libkmsp11.so`[8].

[7] It is recommended to use version 1.7.32 or newer of the Primus HSM PKCS#11 Provider.

[8] Must be used as described in PKCS#11 devices that contain private keys only.

‹SlotId› (optional). If it is not defined, it is searched for the first slot that contains a running token.

‹Pin› (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\siecap11.dll;4;123456"
```

> **Note:** Some PKCS#11 drivers require the `Terminate` method to be called. Otherwise, your application may 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
- Securosys SA CloudsHSM[7]

Selecting a certificate for signing

The 3-Heights® PDF Security 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`
   - SHA-1 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 serial number**
   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 issuer (optional)**
   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, 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. 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 Windows Explorer.
3. Click “Install Certificate...”.
4. Select “automatically select the certificate store based on the type of certificate” and finish import.

PKCS#11 devices that contain private keys only

Some PKCS#11 devices, such as the Google Cloud HSM (Cloud KMS), can only store private keys and no certificates. In such cases, it is possible to supply the required certificates externally using the method `SetSessionProperty`.

Name	Type	Required	Value
Certificate	Bytes	Required	The signing certificate in either PEM (.pem, ASCII text) or DER (.cer, binary) form. This certificate must be selected as the signing certificate as described in Selecting a certificate for signing.
PrivateKeyUri	String	Required	The RFC 7512 URI specifying the private key object in the store. The following URI formats are supported: **`pkcs11:object=‹label›`** To specify the CKA_LABEL object attribute of the private key. The `‹label›` is a text string that is converted to UTF-8 and percent-decoded before matching the CKA_LABEL attribute. Example: `"pkcs11:object=Signing Certificate"` **`pkcs11:id=‹id›`** To specify the CKA_ID object attribute of the private key. The value of the `‹id›` can be percent-encoded to match CKA_ID attributes with binary data. Example: `"pkcs11:id=%C8%48%EC%66%00%17%01%BA%AE%06"` This private key object must belong to the certificate that was specified by the session property **Certificate**.

TrustChain	Bytes	Recommended	The certificates of the trust chain in either PEM (.pem, ASCII text) or DER (.cer, binary) form. Multiple certificates can be concatenated into a single byte stream. Supplying the certificates is highly recommended and required, if revocation information (CRL, OCSP) should be embedded (see property `EmbedRevocationInfo`).

```
using (Secure doc = new Secure())
{
  if (!doc.Open("input.pdf", ""))
    throw new Exception("Error opening input.pdf: " + doc.ErrorMessage);

  doc.SetSessionPropertyBytes("Certificate", File.ReadAllBytes("signing-certificate.cer"));
  doc.SetSessionPropertyString("PrivateKeyUri", "pkcs11:object=Signing Certificate");
  doc.SetSessionPropertyBytes("TrustChain", File.ReadAllBytes("trust-chain.cer"));

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

  using (Signature sig = new Signature())
  {
    sig.Name = "Signing Certificate";
    doc.AddSignature(sig);
  }

  if (!doc.SaveAs("signed.pdf", "", "", PDFPermission.ePermNoEncryption, 0, "", ""))
    throw new Exception("Error saving signed.pdf: " + doc.ErrorMessage);
}
```

5.5.2 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 may use other signing algorithms, specifically the issuer signatures of the trust chain, the timestamp'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.

5.6 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 Security 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 smart card, 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 Security 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. Use of SHA-1 is strongly discouraged by the cryptographic community.

5.6.1 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, semicolumn separated parameter `‹PIN›`. This does not work with qualified certificates, because they always require the PIN code to be entered manually 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” 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 on as the user. 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 Security API is run interactively.

Certificates in the “Local Machine” store 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`) 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.

5.6.2 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. For more information, see Selecting a certificate for signing.

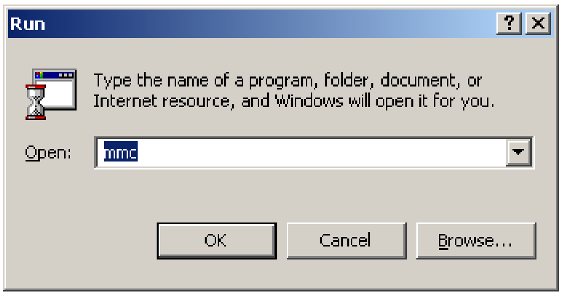
5.6.3 Certificates

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. To see the certificates available on the system, perform the following steps:

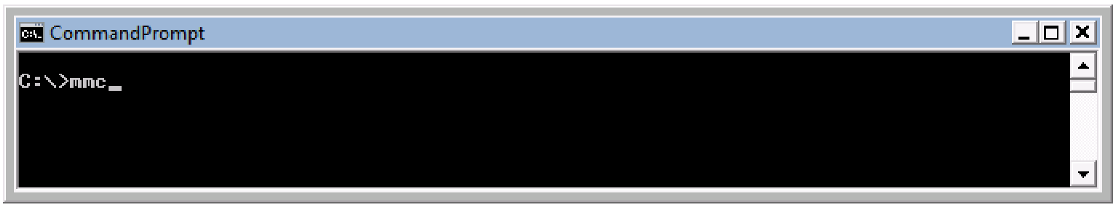
1. To launch the MMC, go to Start → Run…→ type “mmc”, or start a Command Prompt and type “mmc”.



Run

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

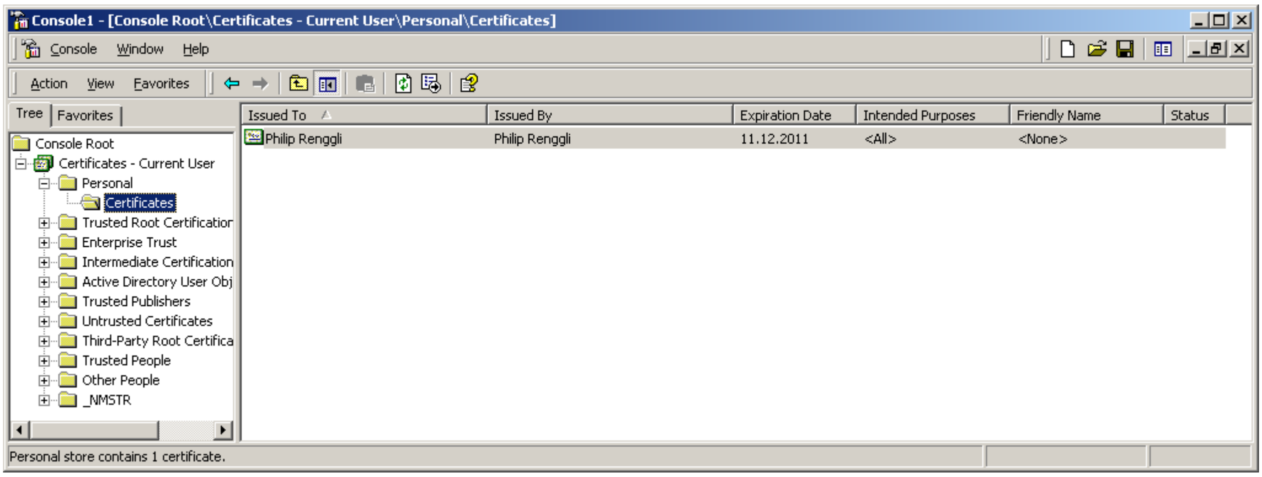
Open: mmc



CommandPrompt

```
C:\>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:

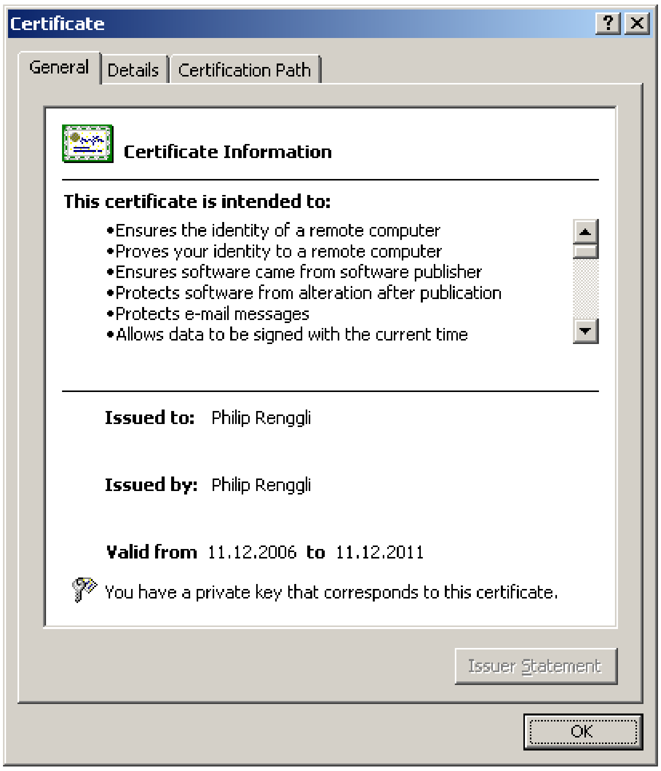


Console1 - [Console Root\Certificates - Current User\Personal\Certificates]

Issued To	Issued By	Expiration Date	Intended Purposes	Friendly Name	Status
Philip Renggli	Philip Renggli	11.12.2011	<All>	<None>	

Personal store contains 1 certificate.

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



Certificate Information

This certificate is intended to:

- Ensures the identity of a remote computer
- Proves your identity to a remote computer
- Ensures software came from software publisher
- Protects software from alteration after publication
- Protects e-mail messages
- Allows data to be signed with the current time

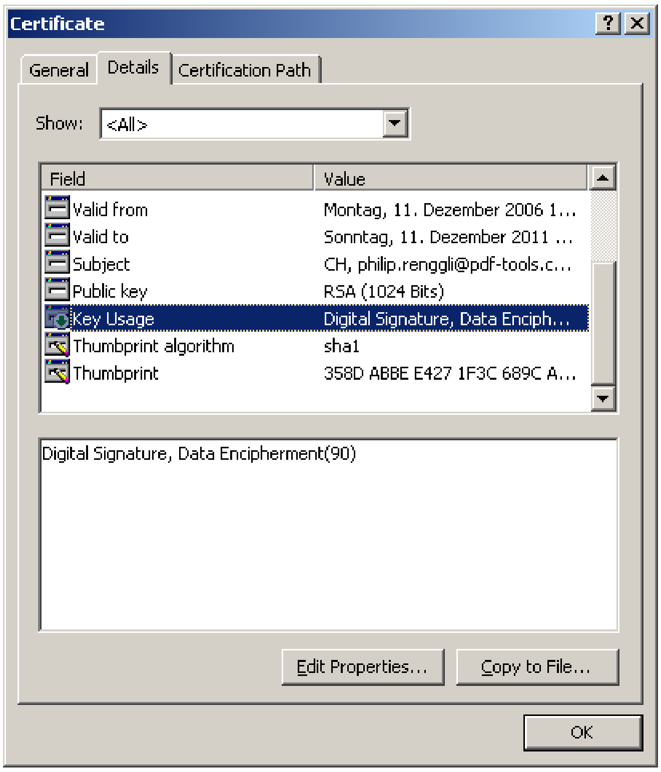
Issued to: Philip Renggli

Issued by: Philip Renggli

Valid from 11.12.2006 **to** 11.12.2011

You have a private key that corresponds to this certificate.

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



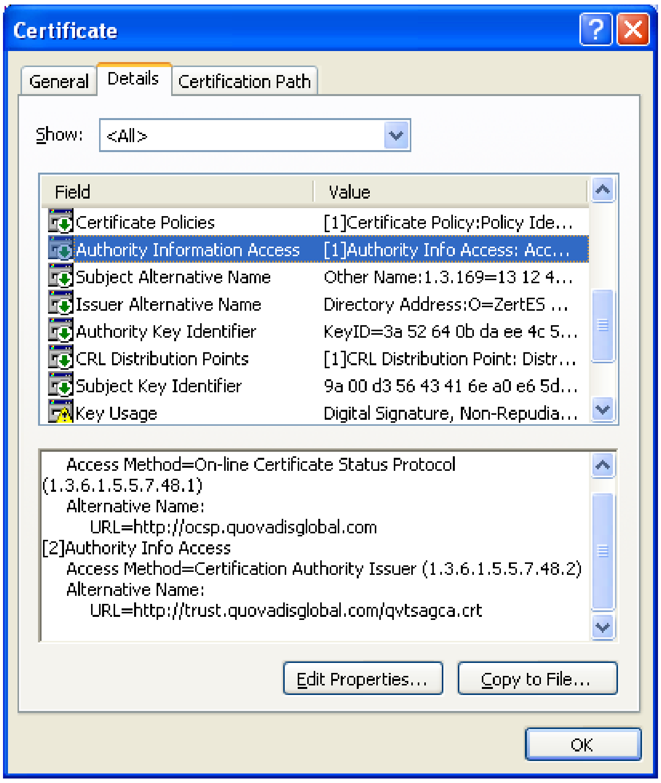
Show: <All>

Field	Value
Valid from	Montag, 11. Dezember 2006 1...
Valid to	Sonntag, 11. Dezember 2011 ...
Subject	CH, philip.renggli@pdf-tools.c...
Public key	RSA (1024 Bits)
Key Usage	Digital Signature, Data Enciph...
Thumbprint algorithm	sha1
Thumbprint	358D ABBE E427 1F3C 689C A...

Digital Signature, Data Encipherment(90)

5.6.4 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.



Show: <All>

Field	Value
Certificate Policies	[1]Certificate Policy:Policy Ide...
Authority Information Access	[1]Authority Info Access: Acc...
Subject Alternative Name	Other Name:1.3.169=13 12 4...
Issuer Alternative Name	Directory Address:O=ZertES ...
Authority Key Identifier	KeyID=3a 52 64 0b da ee 4c 5...
CRL Distribution Points	[1]CRL Distribution Point: Distr...
Subject Key Identifier	9a 00 d3 56 43 41 6e a0 e6 5d...
Key Usage	Digital Signature, Non-Repudia...

```
     Access Method=On-line Certificate Status Protocol
(1.3.6.1.5.5.7.48.1)
     Alternative Name:
          URL=http://ocsp.quovadisglobal.com
[2]Authority Info Access
     Access Method=Certification Authority Issuer (1.3.6.1.5.5.7.48.2)
     Alternative Name:
          URL=http://trust.quovadisglobal.com/qvtsagca.crt
```

5.6.5 Cryptographic suites

The message digest algorithm and 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.

5.7 myBica Digital Signing Service

Provider Property `Provider` or argument of `BeginSession`

The provider configuration string contains the URL to the service endpoint, typically, `https://sign.mybica.ch/DS/DS`.

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://www.pdf-tools.com/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 Security API.

Description	Required	Value	Setting
Common Name	Required	The name of the signer must be set[9].	Property Name.
Timestamp	optional	Use the value urn:ietf:rfc:3161 to embed a timestamp.	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 Creating a visual appearance of a signature.

Proxy configuration If a proxy is used for the connection to the service, see Using a proxy for more information.

5.8 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`
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.

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

Description	Required	Value	Setting
Common Name	Required	The name of the signer must be set[10].	Property Name.
Timestamp	-	Not available.	
Revocation Info	Recommended	To embed OCSP responses or CRL.	Property EmbedRevocationInfo
Visual Appearance	Optional	See Creating a visual appearance of a signature.	

Proxy configuration If a proxy is used for the connection to the service, see Using a proxy for more information.

5.9 Swisscom All-in Signing Service

5.9.1 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[11].	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
Timestamp	optional	Use the value `urn:ietf:rfc:3161` to embed a timestamp.	Property TimeStampURL
Signature Format	Optional	To set the signature format	Property SubFilter. Supported values are `adbe.pkcs7.detached`, `ETSI.CAdES.detached`, `ETSI.RFC3161`[12].
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 Using a proxy for more information.

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

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

[12] `ETSI.RFC3161` is automatically set when signing with AddTimeStampSignature

5.9.2 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›`
RequestID	String	Recommended	Any string that can be used to track the request. Example: An UUID like `AE57F021-C0EB-4AE0-8E5E-67FB93E5BC7F`

5.9.3 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`

5.9.4 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.

5.10 GlobalSign Digital Signing Service

Provider Property `Provider` or argument of i`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: `{}` Example for an account with a dynamic identity: `{ "subject_dn": { "common_name": "John Doe" } }`
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.

Signature configuration The signature can be customized using standard properties of the 3-Heights® PDF Security API.

Description	Required	Value	Setting
Common Name	Required	The name of the signer must be set[13].	Property `Name`.
Timestamp	recommended	Use the value `urn:ietf:rfc:3161` to embed a timestamp.	Property `TimeStampURL`
Signature Format	Optional	To set the signature format	Property `SubFilter`. Supported values are `adbe.pkcs7.detached`, `ETSI.CAdES.detached`, `ETSI.RFC3161`[12].
Revocation Info	Recommended	To embed OCSP responses or CRL.	Property `EmbedRevocationInfo`
Visual Appearance	Optional	See Creating a visual appearance of a signature.	

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

Proxy configuration If a proxy is used for the connection to the service, see Using a proxy for more information.

Creating 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
```

Getting 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.

There are rate limits for both creating new identities and for signing operations. If multiple documents must be signed at once, 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` or `EmbedRevocationInfo`. Furthermore, it is highly recommended to embed a timestamp to prove that the signature was created during the certificate's validity period.

5.11 Custom signature handler

The 3-Heights® PDF Security API provides the capability of replacing the default built-in signature handler with a custom signature handler. A custom signature handler has full control over the creation and validation of the cryptographic part of a signature. This makes it possible to implement proprietary signing algorithms.

The custom signature handler must implement a C interface as described in the header file `pdfsignaturehandler.h`. It can be registered using a call to `PdfRegisterSignatureHandler()` during the initialization of the 3-Heights® PDF Security API. When using a custom signature handler, it is important that this call be made before using the API for signing.

This allows the PDF and signature technologies to be treated separately and also provides an easy way to replace a signature handler.

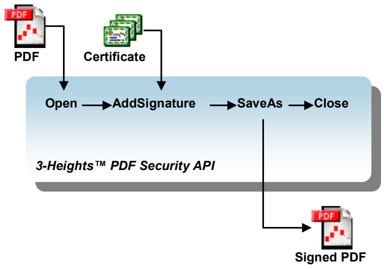
%

6 Creating 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 Creating electronic signatures.

6.1 Signing a PDF document

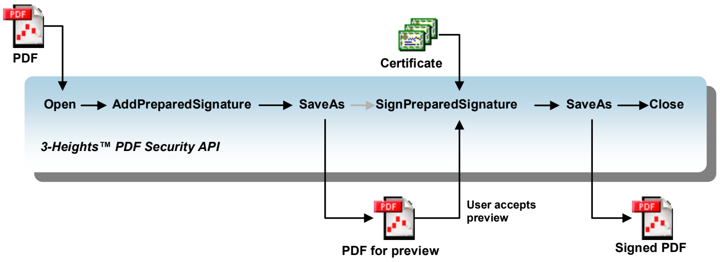
As seen previously in Creating electronic signatures, the process steps to add a signature are as shown in the graphic below:



1. A PDF input document is opened
2. A signature is created and added using a certificate
3. A new, signed PDF output document is created
4. The input document is closed

6.2 Creating a preview of a signed document

The 3-Heights® PDF Security API lets you create a PDF document with a visual appearance of a digital signature without actually signing the document. This document can be used for a preview. If the preview is accepted, the document can be signed without visually changing the document. The process steps to prepare a document for signing and actually sign it upon approval of the user are as shown in the graphic below:



1. A PDF input document is opened.
2. A digital signature is prepared and a visual appearance is generated.
3. A new preview PDF output document is created. This document does not contain a digital signature; however, it contains a placeholder for a signature.
4. If the preview PDF is approved, the document is signed using a certificate.
5. A new, signed PDF output document is created, which looks identical to the preview PDF.
6. The input document is closed.

6.3 Creating a PAdES signature

The PAdES European standard (ETSI EN 319 142) recommends that one of the following four baseline signature levels be used:

PAdES-B-B A digital signature.

PAdES-B-T A digital signature with a timestamp token.

PAdES-B-LT A digital signature with a timestamp token and signature validation data. The signature is a long-term signature or “LTV enabled”.

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

The lifecycle of digital signatures and the usage of these signature levels 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 Security API are compatible.

The Compatibility of PAdES signature levels shows how the signature levels described above and as created by the 3-Heights® PDF Security API conform with other standards.

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 Creating electronic signatures.

Requirements

Level	Signing Certificate	Timestamp	Product
PAdES-B-B	any	no	3-Heights® PDF Security API
PAdES-B-T	any	required	3-Heights® PDF Security API
PAdES-B-LT	advanced or qualified certificate	required	3-Heights® PDF Security API

Requirements

PAdES-B-LTA	advanced or qualified certificate	required	3-Heights® PDF Security API

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 timestamp signature, if your TSA server does not include them in the timestamp.

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 `SaveAs` returns true.

Note on encryption and linearization: Because signature levels PAdES-B-LT and PAdES-B-LTA must be created in a two-step process, the files cannot be linearized and encryption parameters cannot be changed. When creating signature levels PAdES-B-B or PAdES-B-T that may later be augmented, linearization should not be used and all encryption parameters (user password, owner password, permission flags, and encryption algorithm) must be the same for both steps.

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.

6.3.1 Create a PAdES-B-B signature

Input document Any PDF document.

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

```
using (Secure doc = new Secure())
{
  if (!doc.Open("input.pdf", ""))
    throw new Exception("Error opening input.pdf: " + doc.ErrorMessage);

  if (!doc.BeginSession(@"myPKCS11.dll;;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.SaveAs("pades-b-b.pdf", "", "", PDFPermission.ePermNoEncryption, 0, "", ""))
    throw new Exception("Error saving pades-b-b.pdf: " + doc.ErrorMessage);
}
```

6.3.2 Create a PAdES-B-T signature

Input document Any PDF document.

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

```
using (Secure doc = new Secure())
{
  if (!doc.Open("input.pdf", ""))
    throw new Exception("Error opening input.pdf: " + doc.ErrorMessage);

  if (!doc.BeginSession(@"myPKCS11.dll;;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.SaveAs("pades-b-t.pdf", "", "", PDFPermission.ePermNoEncryption, 0, "", ""))
    throw new Exception("Error converting pades-b-t.pdf: " + doc.ErrorMessage);
}
```

6.3.3 Create a PAdES-B-LT signature

Input document A PDF document with a PAdES-B-T signature created using an advanced or qualified certificate.

Cryptographic provider Any cryptographic provider.

```
using (Secure doc = new Secure())
{
  if (!doc.Open("pades-b-t.pdf", ""))
    throw new Exception("Error opening pades-b-t.pdf: " + doc.ErrorMessage);

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

  for (int i = 0; i < doc.SignatureCount; i++)
    using (Signature sig = doc.GetSignature(i))
    {
      if (sig.HasSignature &&
          !doc.AddValidationInformation(sig))
        throw new Exception("Error adding validation infformation to \""
                            + sig.Name + "\": " + doc.ErrorMessage);
    }

  if (!doc.SaveAs("pades-b-lt.pdf", "", "",
                  PDFPermission.ePermNoEncryption, 0, "", ""))
    throw new Exception("Error saving pades-b-lt.pdf: " + doc.ErrorMessage);
}
```

6.3.4 Create a PAdES-B-LTA signature or extend longevity of a signature

Input document

- A PDF document with a PAdES-B-T signature created using an advanced or qualified certificate, or
- a PAdES-B-LTA signature whose longevity should be extended.

Cryptographic provider Any cryptographic provider whose trust store contains all certificates requried for AddValidationInformation.

```
using (Secure doc = new Secure())
{
  if (!doc.Open("pades-b-t.pdf", ""))
    throw new Exception("Error opening pades-b-t.pdf: " + doc.ErrorMessage);

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

  for (int i = 0; i < doc.SignatureCount; i++)
    using (Signature sig = doc.GetSignature(i))
    {
      if (sig.HasSignature &&
          !doc.AddValidationInformation(sig))
        throw new Exception("Error adding validation infformation to \""
                            + sig.Name + "\": " + doc.ErrorMessage);
    }

  using (Signature timeStamp = new Signature())
  {
    timeStamp.TimeStampURL = "http://server.mydomain.com/tsa";
    doc.AddTimeStampSignature(timeStamp);
  }

  if (!doc.SaveAs("pades-b-lta.pdf", "", "",
                  PDFPermission.ePermNoEncryption, 0, "", ""))
    throw new Exception("Error saving pades-b-lta.pdf: " + doc.ErrorMessage);
}
```

6.4 Applying multiple signatures

Multiple signatures can be applied to a PDF document. One signature must be applied at the time. Signing a signed file does not break existing signatures, because the 3-Heights® PDF Security API uses an incremental update.

Signing a linearized file renders the linearization information unusable. Therefore, it is recommended to not linearize files that need to be signed multiple times.

```
using (Secure doc = new Secure())
{
    if (!doc.BeginSession("cvp11.dll;0;secret-pin"))
        throw new Exception("Unable to connect to Cryptographic Provider: "
                            + doc.ErrorMessage);
    if (!doc.Open(inputPath, ""))
        throw new Exception("Document " + inputPath + " cannot be opened: "
                            + doc.ErrorMessage);

    using (Signature signature = new Signature())
    {
        signature.Name = "First Signer";
        doc.AddSignature(signature);
    }

    var tmp = new System.IO.MemoryStream();
    if (!doc.SaveAsStream(tmp, "", "", PDFPermission.ePermNoEncryption, 0, "", ""))
        throw new Exception("Unable to sign temporary document: " + doc.ErrorMessage);

    doc.Close();

    if (!doc.OpenStream(tmp, ""))
        throw new Exception("Temporary document cannot be opened: " + doc.ErrorMessage);

    using (Signature signature = new Signature())
    {
        signature.Name = "Second Signer";
        doc.AddSignature(signature);
    }

    if (!doc.SaveAs(outputPath, "", "", PDFPermission.ePermNoEncryption, 0, "", ""))
        throw new Exception("Unable to sign "+outputPath+": "+doc.ErrorMessage);

    doc.Close();

    doc.EndSession();
}
```

6.5 Creating a timestamp signature

For a timestamp signature, no local signing certificate is required. Instead the timestamp signature requested from the timestamp authority (TSA) is embedded into the document. Nonetheless, a Cryptographic provider that supports timestamp signatures is required.

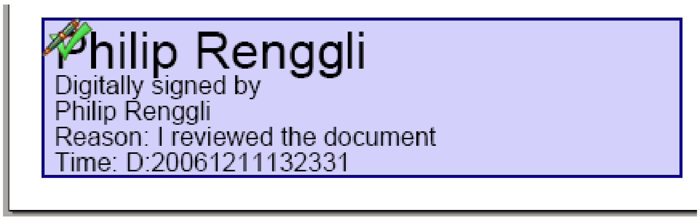
Example: Create a timestamp signature using the method `AddTimeStampSignature`.

```
using (Secure doc = new Secure())
{
  if (!doc.Open("input.pdf", ""))
    throw new Exception("Error opening input.pdf: " + doc.ErrorMessage);

  using (Signature timeStamp = new Signature())
  {
    timeStamp.Provider = "myPKCS11.dll";
    timeStamp.TimeStampURL = "http://server.mydomain.com/tsa";
    doc.AddTimeStampSignature(timeStamp);
  }

  if (!doc.SaveAs("output.pdf", "", "",
                    PDFPermission.ePermNoEncryption, 0, "", ""))
    throw new Exception("Error saving output.pdf: " + doc.ErrorMessage);
}
```

6.6 Creating 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 Security API creates invisible signatures by default.

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:



Philip Renggli
Digitally signed by
Philip Renggli
Reason: I reviewed the document
Time: D:20061211132331

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`, `Text1Color`, `Text2Color`, `FontName1`, `FontName2`, `FontSize1`, and `FontSize2`.

Background image See property `ImageFileName`.

6.7 Guidelines for mass signing

This section provides some guidelines for mass-signing documents using the 3-Heights® PDF Security API.

6.7.1 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 `PdfSecure` object.
2. Open the session to the provider using `BeginSession`.
3. Use the `PdfSecure` object to sign multiple documents.
4. Close the session to the provider using `EndSession`.
5. Dispose of the `PdfSecure` object.

6.7.2 Signing concurrently using multiple threads

The 3-Heights® PDF Security API is thread-safe. Each `PdfSecure` object should be used in one thread at a time only. It is recommended that each thread has a separate `PdfSecure` 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.

6.7.3 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 Security 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");
```

6.8 Miscellaneous

6.8.1 Caching of CRLs, OCSP, and timestamp reponses

To improve the speed when mass signing, the 3-Heights® PDF Security API provides a caching algorithm to store CRL (Certificate Revocation List), OCSP (Online Certificate Status Protocol), TSP (Timestamp 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 Security 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`
Timestamp responses[14]	`<CacheDirectory>/Time Stamps/server.der`

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.

[14] The sizes of the timestamp responses are cached only. Cached timestamp responses cannot be embedded but used for the computation of the signature length only.

6.8.2 Using a proxy

The 3-Heights® PDF Security 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.

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

6.8.3 Configuring a proxy server and firewall

For the application of a timestamp 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.swisssign.net/) via HTTP. The URL for verification is stored in the certificate; the URL for timestamp services is provided by the issuer. If 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

Timestamp

- application/timestamp-query
- application/timestamp-reply

Signature services

- Signature service-specific MIME types.

6.8.4 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 Security API" and its version.

7 Validating digital signatures

7.1 Validating a qualified electronic signature

There are basically three items that need to be validated:

1. Trust chain
2. Revocation information (optional)
3. Timestamp (optional)

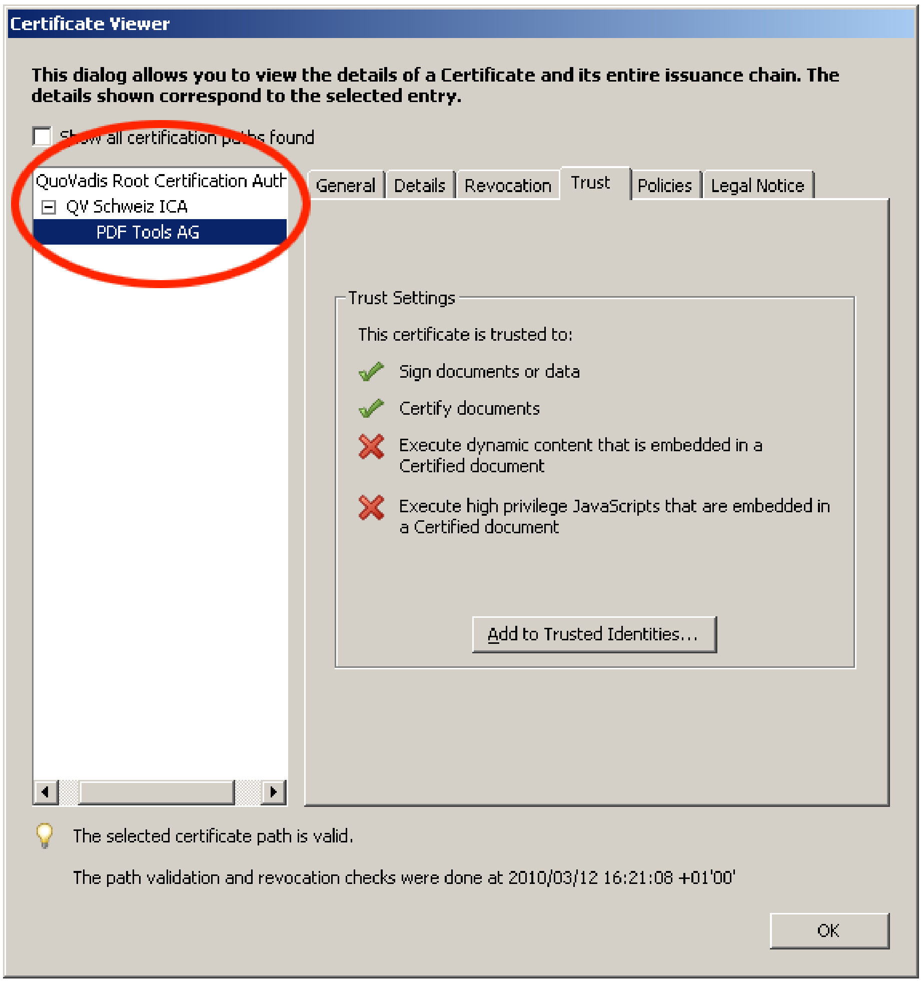
Validation can be done in different ways, e.g. Adobe Acrobat, from which the screenshots below are taken.

7.1.1 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 be 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 open 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”.



Certificate Viewer

This dialog allows you to view the details of a Certificate and its entire issuance chain. The details shown correspond to the selected entry.

□ Show all certification paths found

- QuoVadis Root Certification Auth
  - QV Schweiz ICA
    - PDF Tools AG

Trust Settings

This certificate is trusted to:

✓ Sign documents or data

✓ Certify documents

✗ Execute dynamic content that is embedded in a Certified document

✗ Execute high privilege JavaScripts that are embedded in a Certified document

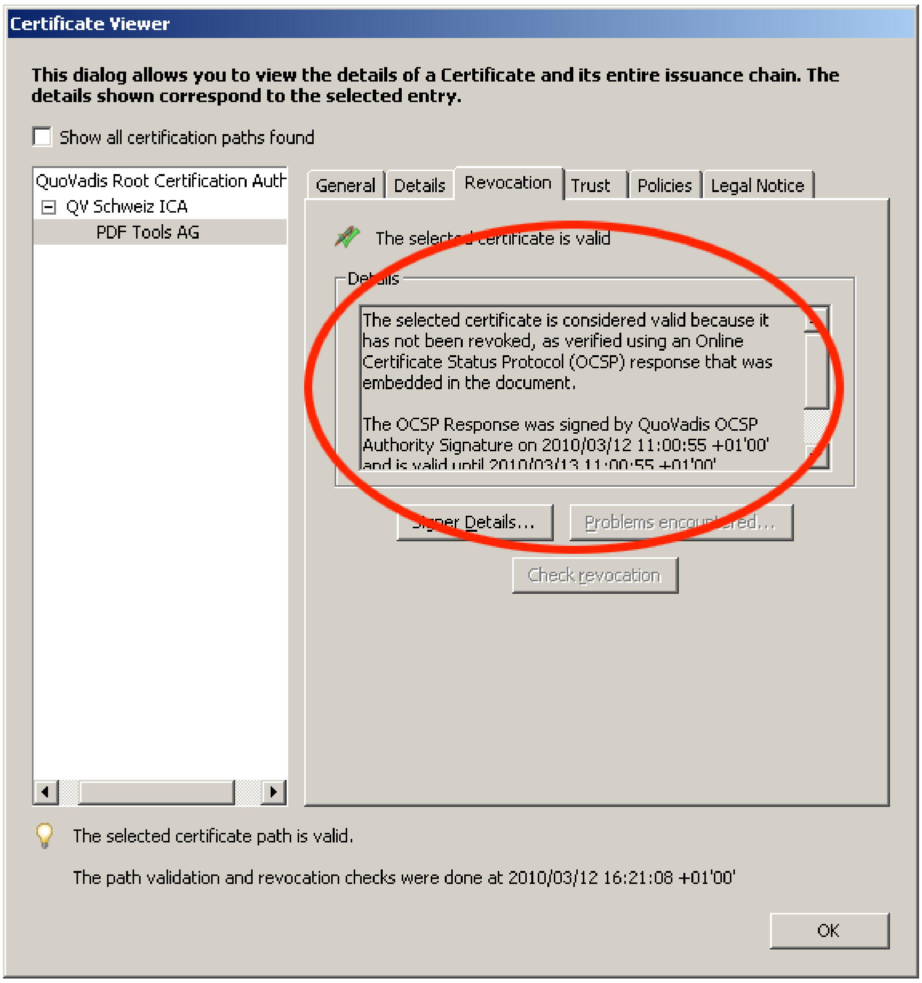
The selected certificate path is valid.

The path validation and revocation checks were done at 2010/03/12 16:21:08 +01'00'

7.1.2 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.



This dialog allows you to view the details of a Certificate and its entire issuance chain. The details shown correspond to the selected entry.

□ Show all certification paths found

QuoVadis Root Certification Auth
⊟ QV Schweiz ICA
PDF Tools AG

The selected certificate is valid

Details

The selected certificate is considered valid because it has not been revoked, as verified using an Online Certificate Status Protocol (OCSP) response that was embedded in the document.

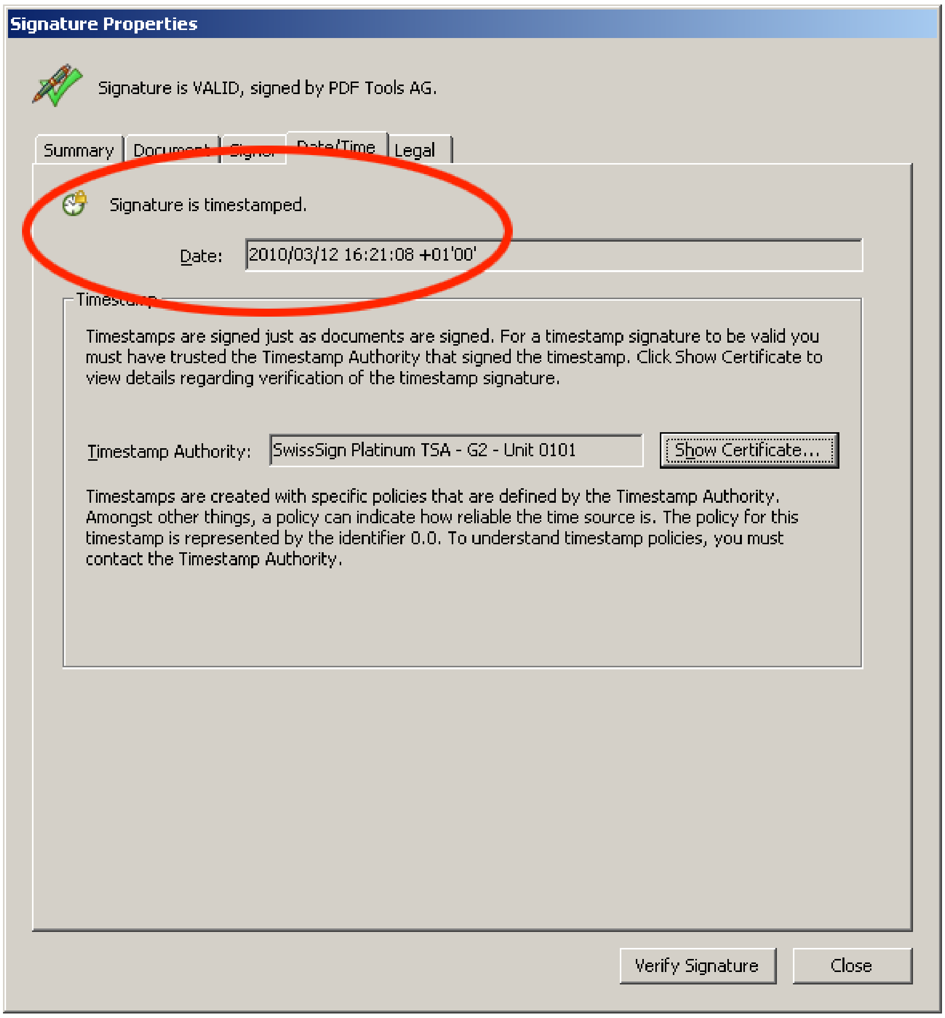
The OCSP Response was signed by QuoVadis OCSP Authority Signature on 2010/03/12 11:00:55 +01'00' and is valid until 2010/03/13 11:00:55 +01'00'

The selected certificate path is valid.

The path validation and revocation checks were done at 2010/03/12 16:21:08 +01'00'

7.1.3 Timestamp

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



Signature is VALID, signed by PDF Tools AG.

Signature is timestamped.

Date: 2010/03/12 16:21:08 +01'00'

Timestamps are signed just as documents are signed. For a timestamp signature to be valid you must have trusted the Timestamp Authority that signed the timestamp. Click Show Certificate to view details regarding verification of the timestamp signature.

Timestamp Authority: SwissSign Platinum TSA - G2 - Unit 0101

Timestamps are created with specific policies that are defined by the Timestamp Authority. Amongst other things, a policy can indicate how reliable the time source is. The policy for this timestamp is represented by the identifier 0.0. To understand timestamp policies, you must contact the Timestamp Authority.

7.2 Validating 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. Timestamp
4. LTV expiration date
5. Other PAdES requirements

7.2.1 Trust chain

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

7.2.2 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.

7.2.3 Timestamp

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

7.2.4 LTV expiration date

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

The lifetime of the protection can be further extended beyond the life of the last timestamp applied by adding further DSS information to validate the previous last timestamp as well as a new timestamp. This process is described in Creating a PAdES signature.

7.2.5 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.

See method `ValidateSignature` in the `PdfSecure` interface.

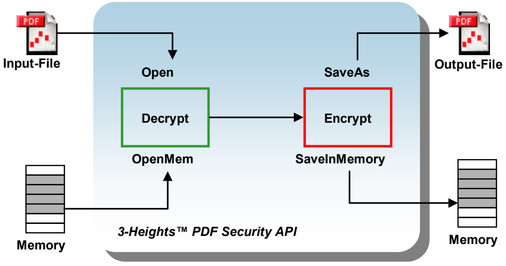
8 Advanced guide

8.1 Using the in-memory functions

The 3-Heights® PDF Security API always requires two PDF documents. A PDF input document from which it reads and a PDF output document to where the result is saved.

To open from and save to files, the Open and SaveAs methods are used. These two methods are described in Encrypting a PDF document and Reading an encrypted PDF document.

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



Once the output document is saved to memory using SaveInMemory, the memory block can be accessed using the GetPdf method.

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

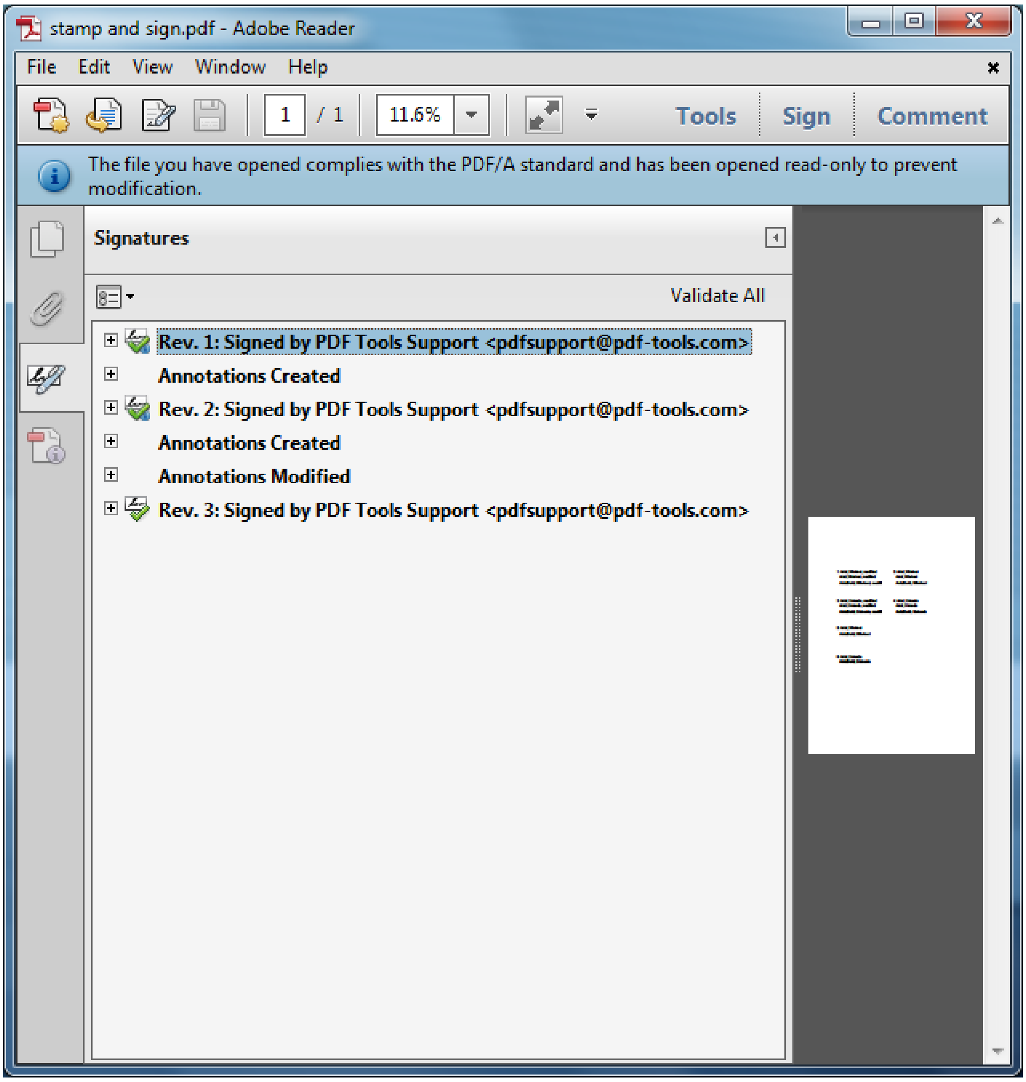
```
PDFSecure1.Open(InputFile)
PDFSecure1.SaveInMemory()
PDFSecure1.Close()
PDFSecure2.OpenMem(PDFSecure1.GetPdf())
PDFSecure2.SaveAs(OutputFile)
PDFSecure2.Close()
```

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

9 Stamping

The 3-Heights® PDF Security API can add new content such as text or images to the output document. This process is called stamping. The content of previously applied stamps can be modified.

The 3-Heights® PDF Security API can sign and stamp documents in one step. To not invalidate existing signatures, stamps can be modified and created using stamp annotations with an incremental update to the input document. An example of this can be seen in the screenshot below.



The file you have opened complies with the PDF/A standard and has been opened read-only to prevent modification.

- Rev. 1: Signed by PDF Tools Support <pdfsupport@pdf-tools.com>
- Annotations Created
- Rev. 2: Signed by PDF Tools Support <pdfsupport@pdf-tools.com>
- Annotations Created
- Annotations Modified
- Rev. 3: Signed by PDF Tools Support <pdfsupport@pdf-tools.com>

9.1 Stamp file syntax

Stamps are described with XML data that is passed to the 3-Heights® PDF Security API either as file using the method `AddStamps, AddStampsMem` or as memory block using the method `AddStamps, AddStampsMem`. A stamp file can contain one or more stamps.

For each `Tag` there is a separate table below, where the `Attribute-Names` and the `Attribute-Values` are described.

`<pdfstamp>`

The **Root Tag** for the PDF stamp XML file. The tag may contain multiple stamps.

`xmlns="http://www.pdf-tools.com/pdfstamp/"` (**required**)
XML namespace used for all stamp elements.

9.1.1 Stamp

A stamp is defined by a `<stamp>` tag that specifies the stamp's size, position, and pages to which it is applied to. The stamp's appearance is defined by the content operators contained therein.

`<stamp>` Add a Stamp

`page="‹page_set›"` (**required**)
The pages to which the stamp is to be applied. The syntax is as follows:

```
‹page_set› = ‹page_range› [","  ‹page_range›]
‹page_range› = ‹n› | ‹n1›-‹n2› | first | last | not_first | not_last | even
| odd | all
```

Where:

- `‹n›`, `‹n1›`, `‹n2›`: Page number. `1` defines the first page.
  The prefix `^` can be used to count from the end of the document. For example, `^1` specifies the last and `^2` the second to last page.
- `first`: First page
- `last`: Last page
- `odd`: Only odd pages including first page and last page in case it is odd
- `even`: Only even pages including last page in case it is even
- `all`: All pages
- `not_first`: First page excluded
- `not_last`: Last page excluded

Example: `page="1,2-4,6,10,last"`

`name="‹identifier›"` (**optional**)
Unique identifier of the stamp, must be less than 127 characters, see section Modify content of existing stamps for more information.

`relativepos="‹x› ‹y›"` (**required**)
Relative position `‹x›` and `‹y›` of the stamp with regards to the page. Positive values of `‹x›` and `‹y›` define the distances of the stamp to the left and lower, negative values to the right and upper page boundary respectively. The units of the values are PDF units of 1/72 inch. The positioning algorithm works best for stamp rotation angles that are a multiple of 90° (see rotate attribute).

`‹x›` or `‹y›` are ignored, if respective `align` is used.

Examples:

1. `relativepos=" 10 -10"` places the stamp in the upper left corner of the page.
2. `relativepos="-10 -10"` places the stamp in the upper right corner of the page.
3. `relativepos=" 10  10"` places the stamp in the lower left corner of the page.
4. `relativepos="-10  10"` places the stamp in the lower right corner of the page.

`align="‹alignment›"` (**optional**)
Align the stamp with the page. Allowed values for `‹alignment›` are:

- `center`: position horizontally at center of page, the `‹x›` value of `relativepos` is ignored.
- `middle`: position vertically at middle of page, the `‹y›` value of `relativepos` is ignored.
- `transverse`: position stamp in the middle of the page and rotate it, such that it aligns with the diagonal of the page from the lower left to the upper right corner. Note that `transverse` cannot be used in combination with the attributes `relativepos` and `rotate`.

Examples:

1. `<stamp position="0 4" align="center">`
   Centers the stamp horizontally and 4 pt away from the bottom of the page.
2. `<stamp position="-4 0" align="middle">`
   Centers the stamp vertically and 4 pt away from the right edge of the page.

`size="‹w› ‹h›"` (**optional**)
The width and height of the stamp. The stamp's content will be clipped to this rectangle. If this is not specified or either `‹w›` or `‹h›` are zero, the respective size is calculated to fit content.

`rotate="‹angle›"` (**optional**)
Rotation of the stamp in degrees clockwise.

`scale="‹scale_set›"` (**optional**)
Modify scale of stamp. Allowed values for `‹scale_set›` are:

- `relToA4`: Scale the stamp relative to the page size. For example, make stamp half as large on an A5 and twice as large on an A3 page as specified.
- `shrinkRelToA4`: Shrink stamp for all pages smaller than A4. For example, on A5 make stamp half as large as specified and as specified an A3 page.

`autoorientation="‹b›"` (**optional**)
Allowed values for `‹b›` are:

- `false` (**default**): Always position stamps as defined by stamp attributes.
- `true`: Detect orientation (portrait and landscape) of page automatically and treat landscape page as 90° rotated portrait. Useful to apply stamps to "long" or "short" edge of page.

`alpha="‹ca›"` (**optional**)
The opacity of the stamp as a whole. `1.0` for fully opaque, `0.0` for fully transparent.

Default: `1.0`

The PDF/A-1 standard does not allow transparency. Therefore, for PDF/A-1 conforming input files you must not set alpha to a value other than `1.0`.

`type="‹type›"` (**optional**)
The type of the stamp

- `annotation` (**default**): The stamp is added to the page as a stamp annotation. Creating or modifying stamps of this type will not invalidate existing signatures of the input document. While it is not easily possible to remove stamps of this type, it is possible to print a document without annotations.
- `foreground`[15]: The stamp is added to the foreground of the page content. Creating or modifying stamps of this type will invalidate all existing signatures of the input document. It is not easily possible to remove stamps of this type nor can the document be printed without them.
- `background`: The stamp is added to the background of the page content. Creating or modifying stamps of this type will invalidate all existing signatures of the input document. It is not easily possible to remove stamps of this type nor can the document be printed without them.
  Note that stamps placed this way can be hidden when pages contain a non-transparent background. In these cases, you may rather want to put the stamps in the foreground, but apply alpha transparency to achieve a result with existing content not covered completely.

`flags="‹flags›"` (**optional**)
Set the flags of the stamp annotation (i.e. stamps with `type="annotation"`). `‹flags›` is a comma separated list of the following values: `NoView`, `Print`, `ReadOnly`, and `Locked`. See chapter 8.4.2 "Annotation Flags" of the PDF Reference 1.7 for a description of the flags.

For PDF/A conformance, the flag Print must be set and NoView must not be set.

Default: Print, ReadOnly, Locked

layer="‹name›" (**optional**)

Set the name of the layer that can be used by the consumer to selectively view or hide the stamp. If the attribute is omitted or its value is empty, no layer is used so the stamp is always visible.

For input documents that already contain a layer of the specified name the document's existing layer is used. Otherwise, a new layer is created. The new layer is visible by default and inserted at the end of the document's list of layers.

Default: no layer

The PDF/A-1 standard does not allow layers. Therefore, for PDF/A-1 conforming input files you must not set the attribute layer. In order to preserve the conformance of PDF/A-1 input documents, the 3-Heights® PDF Security API will not create layers and indicate a stamping warning by setting ErrorCode to PDF_STMP_W_PS.

Coordinates

All coordinate and size values are in PDF units of 1/72 inch (A4 = 595 x 842 points, letter = 612 x 792 points). The origin of the coordinate system is generally the lower left corner of the reference object. For stamps the reference object is the page, for content operators the reference is the stamp rectangle.

Modify content of existing stamps

Setting the name attribute of a stamp allows the stamp's content to be replaced later. If an existing stamp with the same name exists in the input file, its content is replaced as shown in example Example 2: Modify "Simple Stamp". Note that when updating a stamp, its pageset, position and size cannot be changed. Therefore, if you intend to update a stamp, make sure to create it specifying a size that is sufficiently large.

When modifying a stamp, only its content may be changed. All attributes of <stamp> must remain unchanged, in particular page, size and type.

9.1.2 Stamp content

Each stamp contains a number of content operators that define the appearance (i.e. the content) of the stamp. The content operators are applied in the order they appear within <stamp> where each content element is drawn over all previous elements (i.e. increasing z-order).

Text

Stamp text is defined by <text>. All character data (text) therein is stamped:

```
<text font="Arial" size="12">Some text</text>
```

Text fragments can be formatted differently by enclosing them in a <span> element. All text formatting attributes are inherited from the parent element and can be overridden in <span>:

```
<text font="Arial" size="12" >Text with a <span
```

[15] Up to version 4.5.6.0 of the 3-Heights® PDF Security API this type was called content.

```
  font="Arial,Bold">bold</span> and a <span
  color="1 0 0 ">red</span> word.</text>
```

Note that all character data in `<text>` is added, including whitespace such as spaces and line breaks.

`<text>` Add Text

All text formatting attributes described in `<span>` can also be specified in `<text>`.

`position="‹x› ‹y›"` (**optional**)

The position in points within the stamp, e.g. `"200 300"`.

With the default values for `align` (`align="left top"`), `position` defines the top left corner of the text[16].

`align="‹xalign› ‹yalign›"` (**optional**)

Align text at `position` or stamp, if `position` is not set.

Values for horizontal alignment `‹xalign›`:

- `left`: align to the left (**default**)
- `center`: center text
- `right`: align to the right

Values for vertical alignment `‹yalign›`:

- `top`: align to the top (**default**)
- `middle`: align to the middle
- `bottom`: align to the bottom

Examples:

1. `<text align="left bottom" ...>`
   positions the text in the left bottom corner of the stamp.
2. `<text align="left bottom" position="10 10" ...>`:
   align left bottom corner of text to position `"10 10"`.

`format="‹b›"` (**optional**)

Whether or not to enable formatting of variable text. Allowed values for `‹b›` are `true` and `false` (**default**). See Variable text for more information.

`text="‹text›"` (**optional**)

The text that is to be written, e.g. `text="Hello World"`.

Multi-line text is supported by using the newline character `&#10;`, e.g. `text="1st line&#10;2nd line"`.

If the attribute `text` is not specified, the text content of `<text>` is used. So `<text ... text="Hello World"/>` produces the same result as `<text ...>Hello World</text>`.

`<span>` Define Formatting of Text

Example: `<text font="Arial" size="8"><span font="Arial,Bold">Note:</span> Text can be formatted using &lt;span&gt;.</text>`

`color="‹r› ‹g› ‹b›"` (**optional**)

The color as RGB value, where all values must be in the range from `0` to `1`, e.g.:

- Red: `"1 0 0"`
- Green: `"0 1 0"`

[16] Prior to version 4.4.31.0 of the 3-Heights® PDF Security API, `position` specified the origin of the first character. When upgrading, add `0.75*size` to the value of `‹y›`.

- Yellow: `"1 1 0"`
- Black: `"0 0 0"` (**default**)
- Gray: `"0.5 0.5 0.5"`

`font="‹name›"` (**required**)

The TrueType name of the font, e.g. `"Arial"` or `"Times New Roman,Bold"`, or a complete path to the font, e.g. `"C:\Windows\Fonts\Arial.ttf"`.

TrueType names consist of a font family name, which is optionally followed by a comma and style, e.g. `"Verdana,Italic"`. Commonly available styles are `"Bold"`, `"Italic"`, and `"BoldItalic"`. The respective font must be available in any of the font directories (see Fonts).

`size="‹n›"` (**required**)

The font size in points, e.g. `12`.

If set to `0` or `auto`, the size is chosen such that text fits the stamp's size. This is only allowed under these conditions:

- The `<text>` element is not within a transformation operator.
- The stamp has a fixed size. It can either be defined by the attribute `size` or from updating an existing stamp.
- If the text's attribute `position` is set, the position must be inside the stamp's size.

`fontencoding="‹encoding›"` (**optional**)

This attribute is relevant only, if the stamp will be modified later (see section Modify content of existing stamps).

The PDF/A standard demands that all used fonts must be embedded in the PDF. Since fonts with many glyphs can be very large in size (>20MB), unused glyphs are removed prior to embedding. This process is called subsetting. The value `‹encoding›` controls the subsetting and must be one of the following:

- `Unicode`: (**default**) Only the glyphs used by the stamp are embedded. If the stamp is modified, a new font that includes the new glyph set has to be re-embedded. This setting is recommended for stamps that will not be modified later.
- `WinAnsi`: All glyphs required for WinAnsiEncoding are embedded. Hence the text's characters are be limited to this character set. If the content of the stamp is updated, fonts using `WinAnsi` will be reused.

For example, embedding the font Arial with `Unicode` and approximately ten glyphs uses 20KB while Arial with `WinAnsi` (approximately 200 glyphs) uses 53KB of font data.

`mode="‹modes›"` (**optional**)

The attribute `mode` controls the rendering mode of the text.

Allowed values for `‹modes›` are the following or a combination thereof:

- `fill`: (**default**) The text is filled.
- `stroke`: The text's outlines are stroked. The width of the stroke is specified by `linewidth`.

`linewidth="‹f›"` (**optional**)

Set the line width in points, e.g. `1.0` (**default**).

`decoration="‹decorations›"` (**optional**)

The attribute `decoration` can be used to add any of the following text decorations:

- `underline`: A small line is drawn below the text.

`<link>` Create Link

For all text contained within this element, a link is created. Links work best for stamps with `type="foreground"`, but are possible for other types as well.

Example: `<text font="Arial" size="8">© <link uri="https://www.pdf-tools.com/"`
`> Pdftools – PDF Tools AG</link></text>`

`uri="‹uri›"` (**required**)
The URI which is the link target.

`<filltext>` Obsolete tag.
Starting with version 4.9.1.0 of the 3-Heights® PDF Security API the element `<filltext …>` was rendered obsolete by `<text …>`.

`<stroketext>` Obsolete tag.
Starting with version 4.9.1.0 of the 3-Heights® PDF Security API the element `<stroketext …>` was rendered obsolete by `<text mode="stroke" …>`.

Variable text

Variable text such as the current date or the number of pages can be stamped in `<text>`. The feature must be activated by setting `format="true"`.

Variable text elements are of the following form:

`"{‹value›:‹format›}"`

The `‹value›` defines the type of value. `‹format›` is optional and specifies how the value should be formatted. To stamp the `{` character, it must be escaped by duplicating it: `{{`.

String values

`‹value›` The following values are supported:

- `Title`: the document's title
- `Author`: the name of the person who created the document
- `Subject`: the subject of the document
- `Creator`: the original application that created the document
- `Producer`: the application that created the PDF

Example: Stamp the document author.

Text	Result
`Author: {Author}`	Author: Peter Pan

Date values

`‹value›` The following values are supported:

- `UTC`: the current time in UTC
- `LocalTime`: the current local time
- `CreationDate`: the date and time the document was originally created
- `ModDate`: the date and time the document was most recently modified

`‹format›` The default format is a locale-dependent date and time representation. Alternatively a format string as accepted by `strftime()` can be specified.

Example: Stamp the current local time with the default format.

Text	Result
`Received: {LocalTime}`	Received: Thu Aug 23 14:55:02 2001

Example: Stamp the current date.

Text	Result
`Date: {LocalTime:%d. %m. %Y}`	Date: 23. 8. 2011

Number values

`‹value›` The following values are supported:

- `PageNumber`: the page number. Note that when updating the content of an existing stamp as described in Modify content of existing stamps, the new content can only contain `PageNumber` if the exising stamp also used `PageNumber`.
- `PageCount`: the number of pages in the document

Optionally, an offset can be appended to the `‹value›`, where positive offsets start with `+` and negative with `-`. For example `{PageCount+2}` to add or `{PageCount-2}` to subtract 2 from the actual page count.

`‹format›` Optionally a format string as accepted by `printf()` can be specified.

Example: Stamp the page count.

Text	Result
`{{PageCount}} = {PageCount}`	{PageCount} = 10

Example: Stamp the current date and time onto each page's lower right corner.

```
<?xml version="1.0" encoding="utf-8"?>
<pdfstamp xmlns="http://www.pdf-tools.com/pdfstamp/">
    <stamp page="all" relativepos="-10 10">
        <text font="Arial" size="10" format="true">Date: {LocalTime}</text>
    </stamp>
</pdfstamp>
```

Images and geometric shapes

`<image>` Add Image

`rect="‹x› ‹y› ‹w› ‹h›"` (**required**)

The rectangle where the image is to be placed at. `‹x›` and `‹y›` correspond to the location (lower left corner), and `‹w›` and `‹h›` to width and height of the image, e.g. `100 200 50 50`

`src="‹url›"` (**required**)

The URL or path to the file[17], e.g. `C:/pictures/image1.jpg` or `http://www.mydomain.com/image1.jpg`.

[17] Prior to version 4.10.13.0 of the 3-Heights® PDF Security API, this attribute was called `filename`.

`compression="‹value›"` (**optional**)
By default, bitonal images are compressed with `CCITTFax`, continuous tone images with `DCT` and indexed images with `Flate`. To explicitly set the compression, use this property.

Supported values are:

- `Flate`: Flate encoded
- `DCT`: DCT (JPEG) encoded
- `CCITTFax`: CCITT G4 encoded

`<fillrectangle>` Add Filled Rectangle

`rect="‹x› ‹y› ‹w› ‹h›"` (**optional**)
The coordinates and size of the rectangle. If this value is omitted, the rectangle fills the entire area of the stamp.

`color="‹r› ‹g› ‹b›"` (**optional**)
The fill color of the rectangle. The color as RGB value, where all values must be in the range from `0.0` to `1.0`. The default is black: `"0 0 0"`

`alpha="‹ca›"` (**optional**)
The opacity of the rectangle. `1.0` for fully opaque, `0.0` for fully transparent.

Default: `1.0`

The PDF/A-1 standard does not allow transparency. Therefore, for PDF/A-1 conforming input files you must not set alpha to a value other than `1.0`.

`<strokerectangle>` Add Stroked Rectangle

`linewidth="‹f›"` (**optional**)
Set the line width in points, e.g. `1.0` (**default**).

For the following parameter descriptions see `<fillrectangle>`.

`rect="‹x› ‹y› ‹w› ‹h›"`

`color="‹r› ‹g› ‹b›"`

`alpha="‹ca›"`

Transformations

The transform operators apply to stamp content defined within the tag. For example, this can be used to rotate `<text>` or `<image>`.

`<rotate>` Rotation

`angle="‹n›"` (**required**)
Rotate by `‹n›` degrees counter-clockwise, e.g. `90`

`origin="‹x› ‹y›"` (**required**)
Set the origin of the rotation in points, e.g. `100 100`

`<translate>` Coordinate Translation

`offset="‹x› ‹y›"` (**required**)
The `‹x›` (horizontal) and `‹y›` (vertical) offset in points. A translation by `x y` is equal to a transformation by `1 0 0 1 x y`.

`<transform>` **Coordinate Transformation**

`matrix="<a> <b> <c> <d> <x> <y>"` (**required**)
The transformation matrix to scale, rotate, skew, or translate.

Examples:

1. Identity: `1 0 0 1 0 0`
2. Scale by factor 2 (double size): `2 0 0 2 0 0`
3. Translate 50 points to left, 200 up: `1 0 0 1 50 200`
4. Rotate by x: `cos(x) sin(x) -sin(x) cos(x) 0 0`
   For 90° ($= \pi/2$) that is: `0 1 -1 0 0 0`

9.2 Examples

9.2.1 Example 1: Simple stamps

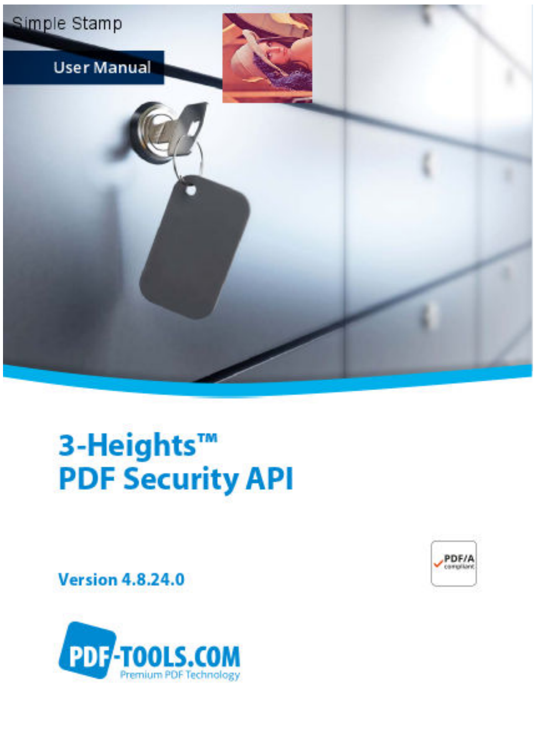
Apply two simple stamps.

First stamp: Stamp text “Simple Stamp” on in upper left corner of all pages.

Second stamp: Stamp image `lena.tif` rotated by 90° and located at the center of the top corner of the first page.

example1.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<pdfstamp xmlns="http://www.pdf-tools.com/pdfstamp/">

  <stamp page="all" name="simple stamp"
   relativepos="10 -10" size="160 0">
     <text align="left middle"
      font="Arial" size="20" fontencoding="WinAnsi"
      text="Simple Stamp" />
  </stamp>

  <stamp page="first"
   relativepos="0 -10" align="center">
    <rotate angle="90" origin="50 50">
      <image rect="0 0 100 100"
       filename="C:\images\lena.tif"/>
    </rotate>
  </stamp>

</pdfstamp>
```



Result of example1.xml.

9.2.2 Example 2: Modify “Simple Stamp”

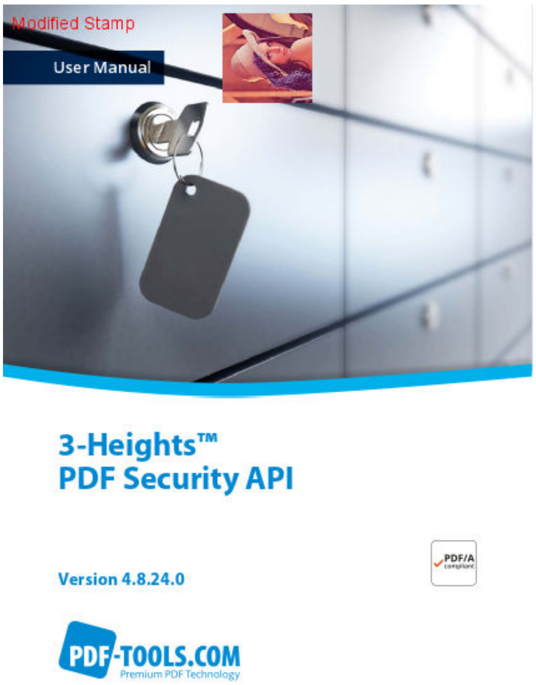
Modify `"simple stamp"` from Example 1: Simple stamps.

The stamp "simple stamp" can be modified by applying the following stamp XML file to the output file of the example above. Note that since position and size of the stamp remain unchanged, the respective attributes can be omitted.

The second stamp applied in Example 1 is not modified.

example2.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<pdfstamp xmlns="http://www.pdf-tools.com/pdfstamp/">
  <stamp name="simple stamp">
    <text align="left middle"
     color="1 0 0"
     font="Arial" size="20" fontencoding="WinAnsi"
     text="Modified Stamp" />
  </stamp>
</pdfstamp>
```


Modified Stamp
User Manual
3-Heights™
PDF Security API
Version 4.8.24.0

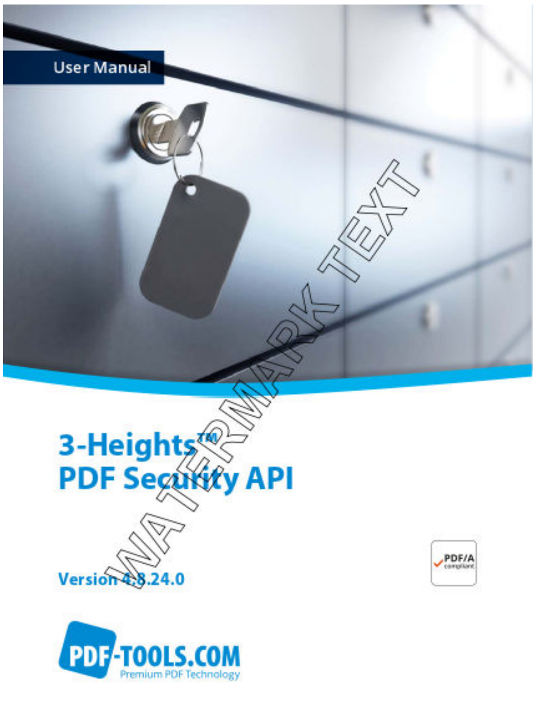
Result of example2.xml.

9.2.3 Example 3: Add watermark text diagonally across pages

The stamp is specified for an A4 page. On each page the stamp is applied to, it is scaled (scale="relToA4") and rotated (align="transverse") to fit the page.

example3.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<pdfstamp xmlns="http://www.pdf-tools.com/pdfstamp/">
  <stamp page="all"
   scale="relToA4"
   align="transverse"
   type="foreground">
    <text mode="stroke"
      font="Arial,Bold" size="60"
      >WATERMARK TEXT</text>
  </stamp>
</pdfstamp>
```



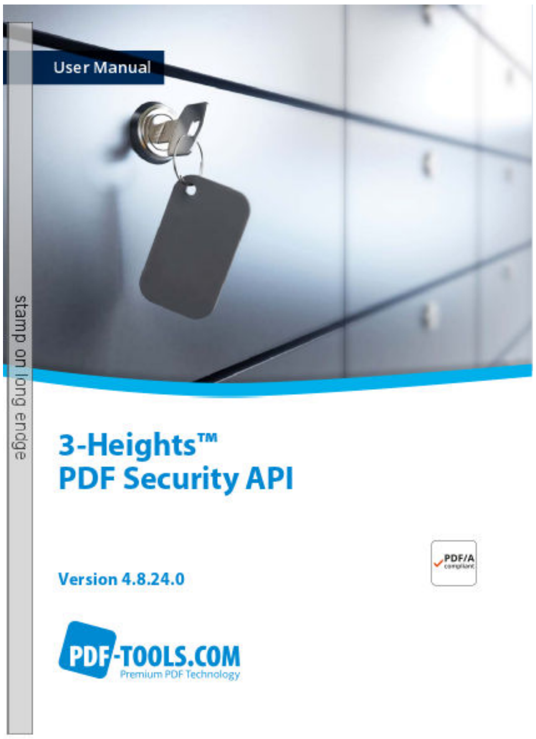
Result of example3.xml.

9.2.4 Example 4: Apply stamp to long edge of all pages

Stamp has a light gray background and a black border.

example4.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<pdfstamp xmlns="http://www.pdf-tools.com/pdfstamp/">
  <stamp page="all" size="802 28"
   relativepos="5 0" align="middle" rotate="90"
   scale="relToA4" autoorientation="true"
   alpha="0.75" type="foreground">
     <fillrectangle color="0.8 0.8 0.8"/>
     <strokerectangle/>
     <text align="center middle"
      font="Arial" size="20"
      text="stamp on long edge"/>
  </stamp>
</pdfstamp>
```



Result of example4.xml.

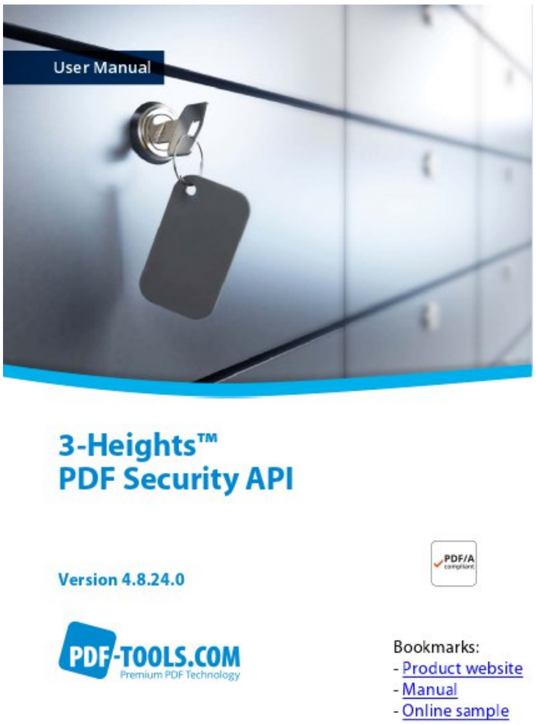
9.2.5 Example 5: Stamp links

Stamp a list of links.

example5.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<pdfstamp xmlns="http://www.pdf-tools.com/pdfstamp/">
  <stamp page="first" type="content" relativepos="-10 10" >
    <text font="MyriadPro" size="20" >Bookmarks:
- <span color="0 0 1" decoration="underline"><link
    uri="http://www.pdf-tools.com/...">Product website</link></span>
- <span color="0 0 1" decoration="underline"><link
    uri="http://www.pdf-tools.com/.../seca.pdf">Manual</link></span>
- <span color="0 0 1" decoration="underline"><link
    uri="https://www.pdf-online.com/osa/secure.aspx">Online sample</link></span>
</text>
  </stamp>
</pdfstamp>
```

Result of example5.xml.



User Manual

3-Heights™
PDF Security API

Version 4.8.24.0

Bookmarks:
- Product website
- Manual
- Online sample

10 Error handling

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

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

Example:

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

Note: When validating signatures using `ValidateSignature`, validation warnings are returned using `ErrorCode`. Therefore, this method is special because `ErrorCode` can be meaningful, even if the method returned `True`. See the method's documentation for a detailed description.

11 Tracing

The 3-Heights® PDF Security API contains tracing functionality that logs runtime information to a file. No confidential data, such as the content of processed files or passwords, are traced. The tracing functionality is designed to provide useful information to Pdftools's support team for support requests. Tracing is not active by default and can be activated by the customer under the guidance of the support team. Nonetheless, activating tracing and sharing the information is optional and there is no obligation to do so.

12 Interface reference

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

12.1 PdfSecure Interface

12.1.1 AddDocMDPSignature

Method: `Boolean AddDocMDPSignature(PdfSignature pSignature, Short accessPermissions)`
License feature: `Signature`

Add a document MDP (modification detection and prevention) signature. A PDF document can, at most, contain one MDP signature. A DocMDP signature defines the access permissions of the document. It should not be combined with standard encryption, i.e. the function `SaveAs` should not apply encryption.

PDF documents with DocMDP signatures added with the 3-Heights® PDF Security API require Acrobat 7 or later to be opened. Since DocMDP signatures were introduced in the PDF Reference 1.6, they cannot be applied to PDF/A-1 input files unless the `ForceSignature` property is set to `True`.

Parameters:

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

`accessPermissions` [`Short`] The access permissions granted are one of the following three values:

1. No changes to the document are permitted; any change to the document invalidates the signature.
2. Permitted changes are filling in forms, instantiating page templates, and signing; other changes invalidate the signature.
3. Permitted changes are the same as for 2, as well as annotation creation, deletion, and modification; other changes invalidate the signature.

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 SaveAs function fails later on.

`False` Otherwise.

12.1.2 AddPreparedSignature

Method: `Boolean AddPreparedSignature(PdfSignature pSignature)`
License feature: `Signature`

Add a signature field including an appearance but without a digital signature. This method must be called prior to `SaveAs` or `SaveInMemory` and should only be used in combination with `SignPreparedSignature`.

Parameter:

`pSignature` [`PdfSignature`] The digital signature from which the field and appearance is created. The properties of the signature must be set before it is added.

Returns:

`True` Successfully prepared signature.

`False` Otherwise.

12.1.3 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 `SaveAs`. Do not dispose of the `PdfSignature` object until the associated document has been saved or closed.

More information on applying digital signatures can be found in Creating 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 `SaveAs` function fails later on.

`False` Otherwise.

12.1.4 AddSignatureField

Method: `Boolean AddSignatureField(PdfSignature pSignature)`
License feature: `Signature`

Add a signature field only. This method adds a field, which is meant to be signed manually in a later step. This method must be called prior to `SaveAs` or `SaveInMemory`.

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 field to the document.

`False` Otherwise.

12.1.5 AddStamps, AddStampsMem

Method: `Boolean AddStamps(String FileName)`
License feature: `Stamping`

Method: `Boolean AddStampsMem(Variant MemBlock)`
License feature: `Stamping`

Add a stamp XML file. This method must be called after the input file is opened and before the save operation. For more information about stamping, see Stamping.

12.1.6 AddTimeStampSignature

Method: `Boolean AddTimeStampSignature(PdfSignature pSignature)`
License feature: `Signature`

Add a document timestamp. The following signature properties must be set: `TimeStampURL`. The following signature properties may be set: `Provider`, `TimeStampCredentials`.

PDF documents with document timestamp signatures require Acrobat X or later to be opened. Since this type of signature was introduced in the PDF 2.0, they cannot be applied to PDF/A-1 input files unless the `ForceSignature` property is set to `True`.

12.1.7 AddValidationInformation

Method: `Boolean AddValidationInformation(PdfSignature pSignature)`
License feature: `Signature`

Add signature validation information to the document security store (DSS). This information includes:

1. All certificates of the signing certificate's trust chain, unless they are already embedded into the signature.
2. Revocation data (OCSP or CRL) for all certificates that support revocation information.

Validation information for embedded timestamp tokens is added as well.

This requires a Cryptographic provider, which has been opened using `BeginSession`. All types of cryptographic providers support this method. However, this method fails when using a provider whose certificate store is missing a required certificate. Because providers of digital signature services do not have a certificate store, it is recommended to use either the PKCS#11 or the Windows Cryptographic provider.

This method can be used to create signatures with long-term validation material or to extend the longevity of existing signatures. See Creating a PAdES signature for more information.

> **Note:** This method does not validate the signature, but only downloads the information required.

> **Note:** Adding validation information for expired certificates is not possible. Therefore, it is crucial to extend the longevity of signatures before they expire.

Parameter:

`pSignature` [`PdfSignature`] The digital signature for which validation information is to be added. This must be an existing signature obtained using `GetSignature` from the currently opened document.

Returns:

`True` Successfully added complete validation information for the signature to the document.

`False` Otherwise.

12.1.8 AutoLinearize

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

> **Note:** With this option enabled, non-Latin characters in the output file name are not supported.

Automatically decide whether to linearize the PDF output file for fast web access.

Applying linearization can lead to a large increase in file size for certain documents. Enabling this option lets the 3-Heights® PDF Security API automatically apply linearization or refrain from doing so based on the estimated file size increase.

With this option enabled, PDF 2.0 documents are automatically excluded from linearization.

See also `Linearize` for more information for linearized PDFs.

Note: If this property is set to True, then the value given to Linearize is ignored.

12.1.9 BeginSession

Method: `Boolean BeginSession(String Provider)`

The BeginSession and EndSession methods support bulk digital signing by keeping the session to the security device (HSM, token or cryptographic provider) open. See 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 Provider property is used for each signature operation.

Parameter:

`Provider` [String] See property Provider.

Returns:

`True` Session started successfully.

`False` Otherwise.

12.1.10 Close

Method: `Boolean Close()`

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

Returns:

`True` The file was closed successfully.

`False` Otherwise.

12.1.11 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 Security API has returned a value, which signales a failure of the function (see Error handling). See also enumeration TPDFErrorCode. Pdftools error codes are listed in the header file bseerror.h. Please note that only few of them are relevant for the 3-Heights® PDF Security API.

12.1.12 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 Security API has returned a value, which signals a failure of the function (see Error handling)

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

.

12.1.13 EndSession

Method: Boolean EndSession()

Ends the open session to the security device.

See BeginSession.

12.1.14 ForceEncryption

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

File encryption is not allowed by the PDF/A standard. Therefore, 3-Heights® PDF Security API aborts and returns an error, when encryption is configured and an input file is PDF/A. Use this option to enable encryption of PDF/A conforming files. The conformance of the output file is downgraded to PDF.

12.1.15 ForceIncrementalUpdate

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

An incremental update is a copy of the original file with all modifications appended to its end. This leaves the original file intact, such that it can later be extracted using GetRevision, GetRevisionFile, GetRevisionStream.

By default, modifications to signed files are performed as incremental updates, which preserves all signatures. Using this property, an incremental update can be forced for other files as well, e.g. in order to preserve external signatures.

When applying an incremental update, all encryption parameters (most importantly the user password) must be the same as in the input file.

Unless a revision is signed, there might be white space characters at the revision's end for which it is unclear to which revision they belong. These white space characters have no influence on the revision's visual appearance or content. However, they may be important in order to preserve external signaures. For a reliable extraction of a revision, it is therefore recommended to save the original file's size. The revision can then be extracted from the updated file by reading all data up to the original file's size.

12.1.16 ForceSignature

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

Force signature allows DocMDP (PDF 1.6) and timestamp signatures (PDF 2.0) on PDF/A-1 documents. The output file's version is upgraded and PDF/A conformance removed. Thus, the output file contains the signature, but is no longer PDF/A-1.

Applying a DocMDP or timestamp signature breaks PDF/A-1 conformance, therefore the default behavior is to abort the operation with an error.

12.1.17 GetPdf

Method: `Variant GetPdf()`

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

Returns:

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

12.1.18 GetRevision, GetRevisionFile, GetRevisionStream

Method: `Variant GetRevision(Integer Revision)`

Method: `Boolean GetRevisionFile(Integer Revision, String FileName)`

Method: `Boolean GetRevisionStream(Integer Revision, Variant Stream)`

Get the PDF document of a given revision number. This is useful to retrieve the state of the PDF document at the time it has been signed. All incremental updates that have been applied after the given revision are ignored.

Parameters:

`Revision` [`Integer`] The revision number (beginning with 0).

`FileName` [`String`] The name of the file to write the revision to.

`Stream` [`Variant`] The stream to write the revision to.

Returns:

The selected revision of the PDF file.

12.1.19 GetMetadata

Method: `Variant GetMetadata()`

Get the the XMP metadata of the input document as byte array. If the document does not contain XMP metadata, `Nothing` is returned.

Returns:

The document XMP metadata as byte array.

12.1.20 GetSignature

Method: `PdfSignature GetSignature(Long iSignature)`
License feature: `Signature`

Get a signature field from the current document.

Parameter:

`iSignature` [`Long`] The selected signature in the document in the range from `0` to ‹n›-1, where 0 is the first and n-1 the last signature. The total number of signatures ‹n› in the document can be retrieved using the `SignatureCount` property.

Returns:

An interface to the PdfSignature.

12.1.21 GetSignatureCount

[Deprecated] Property (get, set): `Long GetSignatureCount`

Use the `SignatureCount` property instead.

12.1.22 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:** 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.

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

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

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

12.1.23 LicenseIsValid

Property (get): `Boolean LicenseIsValid`

Check if the license is valid.

12.1.24 Linearize

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

Note: This property is ignored when AutoLinearize is set to True.

Note: With this option enabled, non-Latin characters in the output file name are not supported.

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

The 3-Heights® PDF Security API does not support linearization of PDF 2.0 documents. For such documents, processing fails. To automatically disable linearization for PDF 2.0, use AutoLinearize.

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.

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

12.1.25 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 Caching of CRLs, OCSP, and timestamp reponses for more information on the caches.

12.1.26 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.

12.1.27 Open

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

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

Parameters:

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

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

Returns:

`True` The file could be successfully opened.

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

12.1.28 OpenMem

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

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

Parameters:

`MemBlock` [`Variant`] The memory block containing the PDF file given as a one-dimensional byte array.

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

Returns:

`True` The document could be successfully opened.

`False` The document could not be opened, it is corrupt, or the password is not valid.

12.1.29 OpenStream

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

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

Parameters:

`Stream` [`Variant`] The stream providing the PDF file. The stream must support random access.

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

Returns:

`True` The document could be successfully opened.

`False` The document could not be opened, it is corrupt, or the password is not valid.

12.1.30 PageCount

Property (get): `Long PageCount`

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

12.1.31 ProductVersion

Property (get): `String ProductVersion`

Get the version of the 3-Heights® PDF Security API in the format "A.C.D.E".

12.1.32 RemoveLegacyStamps

Property (get, set): `Boolean RemoveLegacyStamps`
License feature: `Stamping`

Whether to remove stamps created by the PDF Batch Stamp Tool (`pdstamp`). The stamps must be removable, i.e. they must have previously been added using the `-e` option. After adding removable stamps, the document must not be modified, because this may make the removal of stamps impossible.

Stamps cannot be removed from signed documents, because this would break the signatures. Trying to do so results in an error `PDF_STMP_E_RMLEGACY`. If breaking the signature is acceptable, they must be removed using the `RemoveSignatureField` method.

This option can be used in combination with other operations, e.g. to add new stamps, to sign, or encrypt the result.

12.1.33 RevisionCount

Property (get): `Integer RevisionCount`

Return the number of revisions of the document (the number of incremental updates).

Although a linearized file looks like an incrementally updated file, it only counts as one revision.

See also GetRevision, GetRevisionFile, GetRevisionStream.

12.1.34 RemoveSignatureField

Method: `Boolean RemoveSignatureField(Pdfsignature pSignature)`
License feature: `Signature`

Remove a signature field. An empty signature field can be added using `AddSignatureField`. This method must be called prior to `SaveAs` or `SaveInMemory`.

Removing signature fields breaks the remaining signatures. Therefore, it is important to first remove surplus signatures before signing.

Returns:

`True` Successfully removed the signature field.

`False` Otherwise.

12.1.35 SaveAs

Method: `Boolean SaveAs(String FileName, String UserPw, String OwnerPw, TPDFPermission PermissionFlags, Long KeyLength, String StrF, String StmF)`

Create an output PDF document, apply the security settings and save the content from the input file to the output file.

The last three parameters (`KeyLength`, `StrF`, `StmF`) are only relevant in specific cryptographic situations. In all other cases, it is easiest to use the default values `128`, `"V2"`, `"V2"`.

Parameters:

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

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

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

`PermissionFlags` [`TPDFPermission`] (optional) The permission flags.

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

KeyLength [Long] (optional, Default: 128) The key length is a determining factor of the strength of the encrypting algorithm and the amount of time to break the cryptographic system. For RC4 the key length can be any value from 40 to 128 that is a multiple of 8.

For AESV2 the key length is automatically set to 128, for AESV3 to 256. Notes:

- Certain PDF viewers only support 40 and 128 bit encryption. Other tools, such as the 3-Heights® tools also support other encryption key lengths
- 256 bit encryption requires Acrobat 9 or later.
- If the selected permission flags require a minimum key length, the key length is automatically adjusted (e.g. to 128 bits)

StrF [String] (optional, Default: "V2") Set the string crypt filter. Supported values are "None", "V2", "RC4", "AESV2" and "AESV3". Setting this value to an empty string or Nothing, means the default filter is used. Supported crypt filters are:

- "None": The application does not decrypt data.
- "V2" or "RC4": (PDF 1.2) The application asks the security handler for the encryption key and implicitly decrypts data using the RC4 algorithm.
- "AESV2": (PDF 1.6) The application asks the security handler for the encryption key and implicitly decrypts data using the AES-V2 128 bit algorithm.
- "AESV3": (PDF 1.7) The application asks the security handler for the encryption key and implicitly decrypts data using the AES-V3 256 bit algorithm.

StmF [String] (optional, Default: "V2") Set the stream crypt filter. Supported values are "None", "V2", "RC4", "AESV2" and "AESV3". Note that certain viewers require the stream crypt filter to be equal to the string crypt filter, e.g. both must be RC4 or AES. Setting this value to an empty string or Nothing means the default filter is used.

Returns:

True The opened document could successfully be saved to file.

False Otherwise. One of the following occurred[18]:

- The output file or the signature cannot be created.
- PDF_E_FILECREATE: Failed to create the file.
- SIG_CREA_E_SESSION: Cannot create a session (or CSP).
- SIG_CREA_E_STORE: The certificate store is not available.
- SIG_CREA_E_CERT: The certificate cannot be found.
- SIG_CREA_E_PRIVKEY: The private key is not available.
- SIG_CREA_E_INVCERT: The signing certificate is invalid, because it has expired, is not yet valid, or was revoked.
- 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 timestamp server.
- PDF_E_SIGLENGTH: Incorrect signature length.

Set permission flags equally to Acrobat 7:

In Acrobat 7, there are four different fields/check boxes that can be set. In brackets are the corresponding permission flags.

- Printing Allowed:

[18] This is not a complete list. If SaveAs returns False, it is recommended to abort the processing of the file and log the error code and error message.

  - None ()
  - Low Resolution (ePermPrint)
  - High Resolution (ePermPrint + ePermDigitalPrint)
- Changes Allowed:
  - None ()
  - Inserting, deleting and rotating pages (ePermModify)
  - Filling in form fields and signing existing signature fields (ePermAnnotate)
  - Commenting, filling in form fields, and signing existing signature fields (ePermAnnotate + ePermFillForms)
  - Any except extracting pages (ePermModify + ePermAnnotate + ePermFillForms)
- Enable copying of text, images and other content (ePermCopy + ePermSupportDisabilities)
- Enable text access for screen reader devices for the visually impaired (ePermSupportDisabilities)

These flags can be combined. For example to grant permission which are equal to Acrobat's 7 "Printing Allowed: High Resolution" and "Enable copying of text, images and other content", set the flags ePermPrint + ePermCopy + ePermSupportDisabilities + ePermDigitalPrint.

12.1.36 SaveInMemory

Method: `Boolean SaveInMemory(String UserPw, String OwnerPw, TPDFPermission PermissionFlags, Long KeyLength, String StrF, String StmF)`

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

All parameters are identical to the SaveAs method.

See also Using the in-memory functions.

Returns:

True The document could be saved in memory successfully.

False Otherwise.

12.1.37 SaveAsStream

Method: `Boolean SaveAsStream(Variant Stream, String UserPw, String OwnerPw, Long KeyLength, String StrF, String StmF)`

Parameter:

Stream [Variant] The stream the output file is written to. The stream must support read, write, and random access.

All other parameters and the return value are identical to the SaveAs method.

12.1.38 SetLicenseKey

Method: `Boolean SetLicenseKey(String LicenseKey)`

Sets the license key.

12.1.39 SetMetadata, SetMetadataStream

Method: `Boolean SetMetadata(String FileName)`

Method: `Boolean SetMetadataStream(Variant Stream)`

Set the the XMP metadata of the document.

Parameters:

`FileName` [`String`] The file name where the metadata are read from.

`Stream` [`Variant`] The stream where the metadata are read from.

Returns:

Whether or not the metadata has been set successfully.

12.1.40 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.

12.1.41 SignatureCount

Property (get): `Long SignatureCount`

Return the number of signature fields. If `0` is returned, it means there is no digital signature in the document.

12.1.42 SignPreparedSignature

Method: `Boolean SignPreparedSignature(PdfSignature pSignature)`
License feature: `Signature`

Create a digital signature for an existing signature field, which was previously created using the `AddPreparedSignature` method. This method must be called prior to `SaveAs` or `SaveInMemory`.

Parameter:

`pSignature` [`PdfSignature`] The digital signature that is to be added. This must be the same signature as used in AddPreparedSignature.

Returns:

`True` Successfully added the signature to the document.

`False` Otherwise.

12.1.43 SignSignatureField

Method: `Boolean SignSignatureField(Pdfsignature pSignature)`
License feature: `Signature`

Sign an empty signature field. An empty signature field can be added using `AddSignatureField`. This method must be called prior to `SaveAs` or `SaveInMemory`.

Returns:

`True` Successfully placed the signature into the signature field.

`False` Otherwise.

12.1.44 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 may not be unlocked.

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

12.1.45 TestSession

Method: `Boolean TestSession()`

Test if the current session is still alive.

Returns:

True Subsequent calls to `SaveAs` and `SaveInMemory` are likely to succeed.

False Subsequent calls to `SaveAs` and `SaveInMemory` are unlikely to succeed. Error codes are the same as in `SaveAs`, where applicable.

12.1.46 ValidateSignature

Method: `Boolean ValidateSignature(PdfSignature pSignature)`
License feature: `Signature`

Validate an existing digital signature, which was previously retrieved using the `GetSignature` method. The component supports the verification of signatures including timestamps using cryptographic tokens and hardware security modules (HSM) through their PKCS#11 interface.

The validity checks are carried out at the time indicated either by the embedded timestamp, if present, or by the signing time indicated in the PDF signature field object otherwise. Furthermore, this method extracts the following values from the cryptographic signature and sets the respective properties of the `PdfSignature` object: `Date`, `Email`, `Name`, `Issuer`, `SignerFingerprint`, and `TimeStampFingerprint`.

If you get the error code `SIG_VAL_E_FAILURE`, your cryptographic provider does not offer the algorithms used for the signature. For example, the default provider (CryptoAPI of Windows Cryptographic Provider) does not support the SHA-2 hash algorithms. In this case, choose another provider.

Parameter:

pSignature [`PdfSignature`] The digital signature that is to be validated.

Returns:

True The digital signature is valid, i.e. the document has not been modified. If other problems are detected during signature validation, the `ErrorCode` property may have one of the following values:

1. `SIG_VAL_W_ISSUERCERT`
2. `SIG_VAL_W_TSP`

3. SIG_VAL_W_TSPCERT
4. SIG_VAL_W_NOREVINFO
5. SIG_VAL_W_NOTRUSTCHAIN
6. SIG_VAL_W_TSPNOREVINFO
7. SIG_VAL_W_PADES

The order of the list defines the priority of the error codes from highest to lowest. If multiple problems are detected, the error code with the highest priority is returned.

False The signature is corrupt or the document has been modified.

See also enumeration TPDFErrorCode.

12.2 PdfSignature Interface

This interface lets you create a signature and set its position and appearance. The visual part of the signature consists of two parts. Each part supports multi-line text. The string of both parts are generated automatically based on the signature properties if not set manually.

12.2.1 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.

12.2.2 Contents

Property (get, set): `VARIANT Contents`

Get the Contents of the signature as byte array. This is the actual digital signature, whose format depends on the type of digital signature.

12.2.3 Date

Property (get, set): `String Date`
Default: `"D:00000000000000Z" (set to current date when signature is added)`

This is the date when the signature is added. When this property is not set, the current time and date is used. The format of the date is: `"D:YYYYMMDDHHMMSSZ"`. The meanings are:

D	Header of Date Format
YYYY	year
MM	month
DD	day
HH	hour
MM	minutes
SS	seconds
Z	UTC (Zulu) Time

Example for December 17, 2007, 14:15:13, GMT: `"D:20071217141513Z"`.

> **Note:** This property is set at the time when the signature is applied to the document. If this property is set to an empty string, no entry is created.

12.2.4 DocMdpPermissions

Property (get): `Integer DocMdpPermissions`

Return the document access permissions of a DocMDP signature. For other types of signatures, `0` is returned. See `AddDocMDPSignature` for a description of valid permission values.

12.2.5 DocumentHasBeenModified

Property (get): `Boolean DocumentHasBeenModified`

Get whether the document has been modified (`True`) or not (`False`) since the selected signature was added.

12.2.6 Email

Property (get): `String Email`

This property represents the email address of the signer. The method `ValidateSignature` extracts the address from the signing certificate's subject and sets this property.

12.2.7 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 certificates 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 timestamp signatures. For these cases, `AddValidationInformation` must be used.

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 Caching of CRLs, OCSP, and timestamp reponses.

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 `SaveAs` is `False`, and the `ErrorCode` after `SaveAs` is `SIG_CREA_E_OCSP`.

12.2.8 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.

To avoid setting 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

`color = ‹red› + ‹green›×256 + ‹blue›×256×256,`

where ‹red›, ‹green› and ‹blue› 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`
◼	Gray	`128,128,128`	`8'421'504`
	White	`255,255,255`	`16'777'215`

12.2.9 FieldName

Property (get, set): `String FieldName`

Get or set the name of the signature form field.

If a signature is added to the document and this property is not set, a unique field name is generated.

12.2.10 Filter

Property (get): `String Filter`

Get the name of the preferred signature handler for the signature, such as `"Adobe.PPKLite"`.

12.2.11 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 Fonts.

12.2.12 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`.

12.2.13 Font1Mem

Property (set): `Variant Font1Mem`

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

12.2.14 Font2Mem

Property (set): `Variant Font2Mem`

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

12.2.15 FontSize1

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

Define the font size of the `Text1`.

12.2.16 FontSize2

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

Define the font size of the `Text2`.

12.2.17 HasSignature

Property (get): `Boolean HasSignature`

Get whether the signature has an actual digital signature object or not.

If True, this PdfSignature object can be validated using ValidateSignature. If False, this PdfSignature object can be signed using SignSignatureField.

12.2.18 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 " ".

12.2.19 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 Cryptographic provider).

12.2.20 LineWidth

Property (get, set): Single LineWidth
Default: 2

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

12.2.21 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.

12.2.22 Name

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

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 Digital signatures.

This property can be used to select the signer certificate for signing (see Cryptographic provider).

12.2.23 PageNo

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

Define the page number where the signature is to be added to the document. If an invalid page number is set, it is added to the last page.

The numbers are counted starting from `1` for the first page to the value of `PageCount` for the last page.

12.2.24 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\siecap11.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"
```

12.2.25 ProxyURL

[Deprecated] Property (get, set): `String ProxyURL`
Default: `""`

This property has been deprecated. For more information, see Using a proxy.

12.2.26 ProxyCredentials

[Deprecated] Property (get, set): `String ProxyCredentials`
Default: `""`

This property has been deprecated. For more information, see Using a proxy.

12.2.27 Reason

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

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

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

12.2.28 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]`.

If you are using this property in a programming language that does not support the `Variant` type, to find out what type you should use, create a `PdfSignature` object, and look at the default value of the property in the debugger.

12.2.29 Revision

Property (get): `Integer Revision`

Return the revision number of the PDF document associated with this signature. The associated PDF document can be retrieved using the method `GetRevision`, `GetRevisionFile`, `GetRevisionStream`.

12.2.30 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).

12.2.31 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 Cryptographic provider). After validating a signature, this property contains the validated signature's fingerprint.

12.2.32 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 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". For example, `b5 e4 5c 98 5a 7e 05 ff f4 c6 a3 45 13 48 0b c6 9d e4 5d f5`.

12.2.33 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.

12.2.34 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 Windows Cryptographic Provider.

12.2.35 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 FillColor;

To avoid setting a color, i.e. keep it transparent, set the StrokeColor to -1.

12.2.36 SubFilter

Property (get, set): String SubFilter

Indicates the encoding of the signature. This value is set when extracing signatures using GetSignature and 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) and DocMDP signatures (AddDocMDPSignature).

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) and DocMDP signatures (AddDocMDPSignature). See Creating a PAdES signature for more information.

ETSI.RFC3161 (PDF 2.0) Document timestamp signature (AddTimeStampSignature).

12.2.37 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.

To position text, use the following syntax: `<tab><x>,<y><delimiter><text>`

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

Example: for Visual Basic .NET

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

12.2.38 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 `FillColor`;

12.2.39 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 " ".

12.2.40 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 FillColor;

12.2.41 TimeStampCredentials

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

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

12.2.42 TimeStampFingerprint

Property (get): Variant TimeStampFingerprint

The SHA-1 fingerprint of the timestamp server certificate. After validating a signature that contains a timestamp, this property contains the fingerprint of the timestamp server's certificate.

12.2.43 TimeStampURL

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

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

```
application/timestamp-query
application/timestamp-reply
```

If an invalid timestamp server address is provided or no connection can be made to the time server, the return code of SaveAs is false, and the property ErrorCode is set to SIG_CREA_E_TSP.

12.2.44 UserData

Property (get, set): `Variant UserData`
Default: `Nothing`

This property has only a meaning if a Custom signature handler is used.

12.3 Enumerations

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

12.3.1 TPDFErrorCode Enumeration

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

The single letter gives an indication of the severity of the error. These are: Success, Error, Warning, and Information. In general, an error is returned if an operation could not be completed, 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 header file `bseerror.h`, the javadoc documentation of `com.pdftools.NativeLibrary.ERRORCODE`, and the .NET documentation of `Pdftools.Pdf.PDFErrorCode`. Note that only a few are relevant for the 3-Heights® PDF Security API, most of which are listed here:

TPDFErrorCode table

TPDFErrorCode	Description
`PDF_S_SUCCESS`	The operation was completed successfully.

`LIC_E_NOTSET,`
`LIC_E_NOTFOUND, …`
	Various license management related errors.

`PDF_E_FILEOPEN`	Failed to open the file.
`PDF_E_FILECREATE`	Failed to create the file.
`PDF_E_PASSWORD`	The authentication failed due to a wrong password.
`PDF_E_UNKSECHANDLER`	The file uses a proprietary security handler, e.g. for a proprietary digital rights management (DRM) system.

TPDFErrorCode table

PDF_E_XFANEEDSRENDERING	The file contains unrendered XFA form fields, i.e. the file is an XFA and not a PDF file. The XFA (XML Forms Architecture) specification is referenced as an external document to ISO 32’000-1 (PDF 1.7) and has not yet been standardized by ISO. Technically spoken, an XFA form is included as a resource in a shell PDF. The PDF’s page content is generated dynamically from the XFA data, which is a complex, non-standardized process. For this reason, XFA is forbidden by the ISO Standards ISO 19’005-2 (PDF/A-2) and ISO 32’000-2 (PDF 2.0) and newer.
PDF_W_ENCRYPT	Aborted processing of signed and encrypted document.
PDF_E_PDFASIG	Signature would destroy PDF/A conformance. Signature can be forced using ForceSignature.
PDF_E_INPSIG	Input document must not be signed. Signed input files cannot be linearized, because this would break their signature. Also, the encryption parameters (most importantly the user password) of signed input files cannot be changed.
PDF_STMP_E_PSXML	Invalid stamp xml data.
PDF_STMP_E_PSSTAMP	Invalid stamp description in <ps:stamp>.
PDF_STMP_E_PSOP	Invalid stamp content operator.
PDF_STMP_E_PS	Stamping error. Unable to stamp document. For example, because of a syntax error in the stamp XML file.
PDF_STMP_W_PS	Stamping warning. Document has been stamped, but a warning occurred. For example, no layer has been created in order to preserve PDF/A-1 conformance of input document.
PDF_STMP_E_RMLEGACY	Error removing legacy stamps (see about[prop:RemoveLegacyStamps]).
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 timestamp server.

TPDFErrorCode table

SIG_CREA_E_PRIVKEY	Private key not available. This is usually because a PIN is required and was not entered correctly. 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. Finally, this error could be the result of choosing a message digest algorithm or signing algorithm which is not supported by the provider. See section Cryptographic provider for more information.
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.
PDF_E_SIGLENGTH	Incorrect signature length. 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.
PDF_E_SIGABG	Unable to open signature background image.
PDF_W_NOENCRYPTION	The file is PDF/A and must not be encrypted. Encryption can be forced using `ForceEncryption`.

Validation specific error codes

TPDFErrorCode	Description
SIG_VAL_E_ALGO	Unsupported algorithm found.
SIG_VAL_E_FAILURE	Program failure occurred.
SIG_VAL_E_CMS	Malformed cryptographic message syntax (CMS).
SIG_VAL_E_DIGEST	Digest mismatch (document has been modified).
SIG_VAL_E_SIGNERCERT	Signer’s certificate is missing.
SIG_VAL_E_SIGNATURE	Signature is not valid.
SIG_VAL_W_ISSUERCERT	None of the certificates was found in the store.
SIG_VAL_W_NOTRUSTCHAIN	The trust chain is not embedded.

SIG_VAL_W_TSP	The timestamp is invalid.
SIG_VAL_W_TSPCERT	The timestamp certificate was not found in the store.
SIG_VAL_W_PADES	The signature does not conform to the PAdES standard, e.g. because the signature is not DER encoded or the CMS contains more than one SignerInfo.[19]
SIG_VAL_W_NOREVINFO	Revocation data (OCSP or CRL) is missing for certificate that supports revocation information.
SIG_VAL_E_NOREVINFO	Revocation data (OCSP or CRL) is missing for certificate that supports revocation information.
SIG_VAL_W_TSPNOREVINFO	Revocation data (OCSP or CRL) is missing for certificate in timestamp.
SIG_VAL_E_INVCERT	Invalid certificate, e.g. because it has been revoked or is expired.
SIG_VAL_E_MISSINGCERT	A certificate required for the operation is missing from the certificate store.

12.3.2 TPDFPermission Enumeration

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

TPDFPermission table

TPDFPermissionFlag	Description
ePermNoEncryption	Do not apply encryption. This enumeration value cannot be combined with other values. When using this enumeration, set both passwords to an empty string or Nothing.
ePermSameAsInput	Use the same permissions as present in the input file. This enumeration value cannot be combined with other values.
ePermNone	Grant no permissions
ePermPrint	Low resolution printing
ePermModify	Changing the document
ePermCopy	Content copying or extraction
ePermAnnotate	Annotations
ePermFillForms	Filling of form fields
ePermSupportDisabilities	Support for disabilities

[19] Adobe Acrobat XI classifies such signatures as valid.

TPDFPermission table

ePermAssemble	Document assembly
ePermDigitalPrint	High resolution printing
ePermAll	Grant all permissions

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

> **Note:** The special values ePermSameAsInput and ePermNoEncryption cannot be combined with any other values.

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

Allow Printing

```
Permission = Permission Or ePermPrint
```

Prohibit Printing

```
Permission = Permission And Not ePermPrint
```

13 Version history

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

13.1 Changes in versions 6.19–6.27

- **New** support for Elliptic Curve DSA (ECDSA) signature algorithms in the PKCS#11 Provider and Windows Cryptographic Provider.
- **New** support for PKCS#11 devices that contain private keys only.
- **Update** license agreement to version 2.9
- **Improved** auto font size feature for text stamping feature.

13.2 Changes in versions 6.13–6.18

- Stamping
  - **New** attribute `layer` of `<stamp>` to create stamp whose visiblity can be controlled by layer.
  - **New** value `transverse` for attribute `align` of `<stamp>`.
  - **New** variable text element `PageNumber`.
  - **New** variable text elements for document metadata properties, e.g. `Author` or `Title`.
  - **New** prefix ^ for page numbers in attribute `page` of `<stamp>` to count from back of document.

Interface PdfSecure

- **New** property NoDSS.

13.3 Changes in versions 6.1–6.12

- 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.CAdES.detached`).
  - **Improved** embedding of revocation information (OCSP, CRL, and trust chain) to always use the document security store (DSS)[20].
  - **Changed** the creation of signatures of format `ETSI.CAdES.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`.
  - [C] **Changed** API of the custom signature handler `pdfsignaturehandler.h`.
- Stamping
  - **New** value `shrinkRelToA4` for attribute `flags` of `<stamp>`.
- **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.

[20] Use the property NoDSS to restore the previous behavior.

- [.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 `PdfSecure`

- **New** property `RemoveLegacyStamps` to remove stamps created by the PDF Batch Stamp Tool.

13.4 Changes in version 5

- Digital Signatures
  - **New** support to get CRLs using HTTPS and via HTTP redirection.
- **New** additional supported operating system: Windows Server 2019.
- [PHP] **New** extension PHP 7.3 (non thread safe) for Linux.

Interface `PdfSignature`

- **New** properties `Text1Color` and `Text2Color` to set the color of the signature appearance's text.

13.5 Changes in version 4.12

- **Introduced** license features `Signature` and `Stamping`.
- 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 encryption according to PDF 2.0 (revision 6, replaces deprecated revision 5).
- **Improved** reading and recovery of corrupt TIFF images.
- **New** HTTP proxy setting in the GUI license manager.
- [.NET, C, COM, Java, PHP] **New** property `AutoLinearize` to automatically choose whether to linearize the output document or not.

13.6 Changes in version 4.11

- Digital Signatures
  - **New** support to create Document TimeStamp signatures using Swisscom All-in Signing Service.
  - **New** ability to sign documents that are larger than 2GB (64-bit version only).
- Stamping
  - **New** default compression Flate for PNG images.
- **New** support for reading and writing PDF 2.0 documents.
- **New** support for the creation of output files larger than 10GB (not PDF/A-1).
- **Improved** search in installed font collection to also find fonts by other names than TrueType or PostScript names.

- **New** treatment of the DocumentID. In contrast to the InstanceID the DocumentID of the output document is inherited from the input document.
- [.NET, C, COM, Java, PHP] **Changed** enum TPDFPermission: Added a new value ePermSameAsInput to adopt the encryption parameters from the input document.

Interface PdfSecure

- **New** property DocMdpPermissions: Return the document access permissions of a DocMDP signature.
- [PHP] **Removed** the method Terminate: It is now called automatically by the "PdfTools" PHP extension and has thereby been rendered obsolete.

13.7 Changes in version 4.10

- 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 timestamps.
  - **New** signature algorithm RSA with SSA-PSS (PKCS#1v2.1) can be chosen by setting the provider session property SigAlgo.
  - **Improved** signature validation.
    - More signature formats supported, most notably the new European PAdES norm. The Windows cryptographic provider now supports the same formats as the PKCS#11 provider.
    - Support signature algorithm RSA with SSA-PSS (PKCS#1v2.1).
    - New and improved validation warnings.
    - Check for missing revocation information.
    - Use validation data embedded in the document security store (DSS).
  - **New** ability to add multiple signatures to encrypted files.
- Stamping
  - **New** attribute flags of <stamp>, e.g. to create modifiable stamps or stamps that are only visible when printing.
  - **New** attribute src of <image> allows a HTTP URL or file path.
  - **New** ability to add or modify stamps of signed files that are also encrypted.
- **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.
- [C] **Clarified** Error handling of TPdfStreamDescriptor functions.
- [PHP] **New** Interface for Windows and Linux. Supported versions are PHP 5.6 & 7.0 (Non Thread Safe). The PdfSecureAPI 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.

Interface PdfSecure

- **New** method AddValidationInformation(): Add signature validation information to the document. This method can be used to create signatures with long-term validation material or to enlarge the longevity of existing signatures.
- **Changed** method ValidateSignature():

    - The warning SIG_VAL_W_NOTSP has been removed because it is unnecessary and masks other warnings that have a lower priority. The property TimeStampFingerprint can be used to detect whether a time-stamp is available.
    - See documentation of the method for a list of new warnings.
- [C] **Changed** API of the custom signature handler pdfsignaturehandler.h.

13.8 Changes in version 4.9

- **Improved** behavior: Before signing, missing appearance streams of form fields are created, because otherwise Adobe® Acrobat® cannot validate the signature.
- Stamping:
    - **New** tag <link> to add interactive web links.
    - **New** tag <text> allows to format spans in continuous text using nested <span> tags.
- **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[21].

Interface PdfSecure

- [.NET] **New** methods OpenStream() and SaveAsStream().
- [.NET, C, Java] **New** methods GetRevisionFile() and GetRevisionStream().
- [.NET, C, COM, Java] **New** property ForceIncrementalUpdate.

13.9 Changes in version 4.8

- **New** feature: Images used as signature appearance background or for stamping for PDF/A input files may now have any color space, even if it differs from the input file's output intent.
- **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 PdfSecure

- [.NET, C, COM, Java] **New** property ProductVersion to identify the product version.
- [.NET] **Deprecated** method GetLicenseIsValid.
- [.NET] **New** property LicenseIsValid.

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

14 Licensing, copyright, and contact

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

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

Licensing and copyright The 3-Heights® PDF Security API is copyrighted. This user manual is also copyright protected; It may be copied and distributed provided that it remains unchanged including the copyright notice.

Contact

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