User Manual

3-Heights™ PDF Security API

PDF/A compliant

Version 4.10

PDF-TOOLS.COM
Premium PDF Technology

Contents

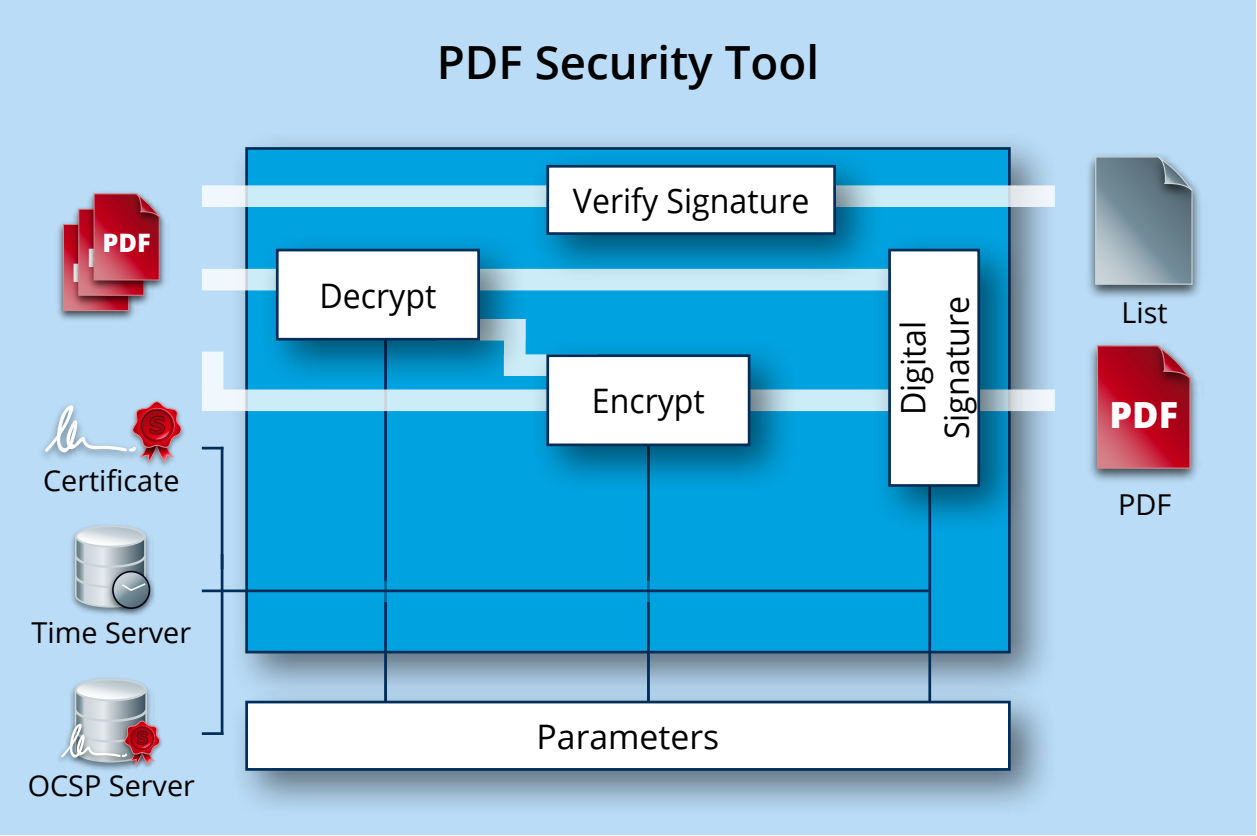
1 Introduction

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 (“smart cards”, “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, PHP) and thanks to its flexibility a developer 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. It can, for instance, 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 compliant signatures
  - Support European Signature Norms
  - Signature types
    - Document signatures to “digitally sign” documents
    - Modification detection & prevention (MDP) signatures to “certify” documents
    - Document time-stamp signatures to “time-stamp” documents
  - Apply PAdES-B-LTA (long term availability and integrity of validation material) and PAdES-LTV (Long Term Validation) signatures
    - Embedded trust chain, time-stamp, and revocation information (OCSP, CRL)
    - Enlarge 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), smart cards, and USB tokens
    - Online signature services
      - 3-Heights™ Signature Creation and Validation Service
      - SwissSign Digital Signing Service
      - SwissSign SuisseID Signing Service
      - QuoVadis sealsign
      - Swisscom All-in Signing Service
      - GlobalSign Digital Signing Service
    - Custom signature handler plugin interface
  - Mass signing of documents
- 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
    - Content extraction 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

    - PDF/A compliant stamps
    - Modify existing stamps
    - Stamping of signed documents preserves existing signatures
- Set document metadata
- Optimize for the web (linearize)
- Read input from and write output document to file, memory, or stream

1.2.2 Formats

Input Formats

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

Target Formats

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

1.2.3 Compliance

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 (ETSI TS 103 172) Part 2 and Part 4 (Long Term Validation, LTV)
- Cryptographic Suites (ETSI TS 119 312)

1.3 Interfaces

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

1.4 Operating Systems

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

- Windows 7, 8, 8.1, 10 – 32 and 64 bit
- Windows Server 2008, 2008 R2, 2012, 2012 R2, 2016 – 32 and 64 bit
- HP-UX 11i and later PA-RISC2.0 – 32 bit
- HP-UX 11i and later ia64 (Itanium) – 64 bit
- IBM AIX 6.1 and later – 64 bit
- Linux 2.6 – 32 and 64 bit
- Oracle Solaris 2.8 and later, SPARC and Intel
- FreeBSD 4.7 and later (32 bit) or FreeBSD 9.3 and later (64 bit, on request)
- macOS 10.4 and later – 32 and 64 bit

1.5 How to Best Read this Manual

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

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

1.6 Digital Signatures

1.6.1 Overview

Digital signature is a large and slightly complex topic. This manual gives an introduction to digital signatures and describes how the 3-Heights™ PDF 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

Abbreviations

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

1.6.3 Why Digitally Signing?

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

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

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

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

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

This part of the signature is entirely created by the 3-Heights™ PDF 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 will fail and will report that the document has been modified since the signature was applied. Only the owner of the certificate and its private key is able to sign the document. However, anybody can verify the signature with the public key contained in the certificate.

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

The 3-Heights™ PDF Security API supports the following types of digital signatures:

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

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

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

1.6.4 What is an Electronic Signature?

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

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

Three types of electronic signatures are distinguished:

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

All applied digital signatures are PDF/A and PAdES compliant.

Simple Electronic Signature

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

Example:

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

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

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

Advanced Electronic Signature

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

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

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

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

This brings the following advantages over a simple electronic signature:

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

Qualified Electronic Signature

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

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

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

This brings the following advantages over an advanced electronic signature:

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

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

1.6.5 How to Create Electronic Signatures

This is a simple example of how to create an electronic document signature. More detailed examples and examples for other types of electronic signatures can be found in How to Create Digital Signatures.

Preparation Steps

1. Identify whether an advanced or a qualified signature is required. For most automated processes an advanced signature is sufficient.
2. Acquire a corresponding certificate from a CA. Note that some CA offer USB sticks or smart cards that contain both, an advanced and a qualified certificate.
3. Setup and configure the certificate's Cryptographic Provider.
   - In case the certificate resides on hardware such as an USB token or a Smart Card, the required middleware (driver) needs to be installed.
   - In case the certificate is a soft certificate, it must be imported into the certificate store of a cryptographic provider.
4. Identify regulatory requirements regarding the content and life cycle of the signature:
   - Is a time-stamp required to prove that the signature itself existed at a certain date and time?
   - Should validation information be embedded, in order to allow the signature to be validated long time after its generation?
   - Should the integrity of the validation material be protected?

   These requirements (or regulatory requirements) define the signature level that must be used.
5. Optional: Acquire access to a trusted time server (TSA) (e.g. from the CA of your signing certificate).
6. Optional: Ensure your input documents conform to the PDF/A standard. It is recommended to sign PDF/A documents only, because this ensures that the file's visual appearance is well defined, such than it can be reproduced flawlessly and authentically in any environment. Furthermore, PDF/A compliance is typically required if the file is to be archived. Because signed files cannot be converted to PDF/A without breaking its signatures, files must be converted before signing.

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

Application of the Signature

Apply the signature by providing the following information:

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

Example: Steps to Add an Electronic Document Signature

The 3-Heights™ PDF Security API applies PDF/A compliant signatures. This means if a PDF/A document is digitally signed, it remains PDF/A compliant.

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

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

In Visual Basic the four steps above look like this:

```
Dim Document As New PDFSECUREAPILib.PdfSecure
Document.Open "input.pdf", ""

Dim Signature As New PDFSECUREAPILib.PdfSignature
Signature.Name = "Philip Renggli"
Signature.Provider = "cvp11.dll;0;secret-pin"
Signature.Reason = "I reviewed the document" ' optional
Signature.Rect = Array(10, 10, 210, 60) ' optional visual appearance
Signature.TimeStampURL = "http://server.mydomain.com:80/tsa" ' optional

Document.AddSignature Signature
Document.SaveAs "output.pdf"
Document.Close
```

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

2 Installation and Deployment

2.1 Windows

The 3-Heights™ PDF Security API comes as a ZIP archive.

The installation of the software requires the following steps.

1. You need administrator rights to install this software.
2. Log in to your download account at http://www.pdf-tools.com. Select the product "PDF Security API". If you have no active downloads available or cannot log in, please contact pdfsales@pdf-tools.com for assistance.
   You will find different versions of the product available. We suggest to download the version, which is selected by default. If another is required, it can be selected using the combo box.
   The product comes as a ZIP archive containing all files.
   There are 32 and 64-bit versions 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 file contains both the 32-bit and the 64-bit version of the product.
3. Unzip the archive to a local folder, e.g. `C:\Program Files\PDF Tools AG\`.
   This creates the following subdirectories:

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

4. (Optional) Register your license key using the License Management.
5. Identify which interface you are using. Perform the specific installation steps for that interface described in chapter Interface Specific Installation Steps
6. Ensure the cache directory exists as described in chapter Special Directories.
7. If you want to sign documents, proceed with setting up your cryptographic provider as described in chapter Cryptographic Provider.
8. If you want to stamp text, proceed with setting the fonts required as described in chapter Fonts.

2.2 Unix

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

The Unix version of the 3-Heights™ PDF Security API provides three interfaces:

- Java interface
- Native C interface
- PHP interface - Linux only

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

File Description

Name	Description
bin/‹platform›/libPdfSecureAPI.so	This is the shared library that contains the main functionality. The file's extension varies depending on the type of UNIX system. The directory ‹platform› is either x86 containing the 32-bit version of the library, or x64 for the 64-bit version.
bin/‹platform›/php*_pdftools.so	The PdfTools PHP extension.
doc/*.*	Documentation
include/*.h	Contains header files to include in your C/C++ project.
jar/SECA.jar	Java API archive.
samples	Example code.

2.2.1 All Unix Platforms

1. Unpack the archive in an installation directory, e.g. /opt/pdf-tools.com/
2. Copy or link the shared object into one of the standard library directories, e.g:

```
ln -s /opt/pdf-tools.com/bin/‹platform›/libPdfSecureAPI.so /usr/lib
```

3. Verify that the GNU shared libraries required by the product are available on your system:
   - *On Linux*:

     ```
     ldd libPdfSecureAPI.so
     ```

   - *On AIX*:

     ```
     dump -H libPdfSecureAPI.so
     ```

   In case the above reports any missing libraries you have two options:
   a. Use your system's package manager to install the missing libraries. On Linux it usually suffices to install the package libstdc++6.
   b. Use the PDF-Tools provided GNU shared libraries:
      1. Go to http://www.pdf-tools.com and navigate to "Support" →"Utilities".
      2. Download the GNU shared libraries for your platform.
      3. Extract the archive and copy or link the libraries into your library directory, e.g /usr/lib or /usr/lib64.
      4. Verify that the GNU shared libraries required by the product are available on your system now.
4. Optionally register your license key using the Command Line License Manager Tool.
5. Identify which interface you are using. Perform the specific installation steps for that interface described in chapter Interface Specific Installation Steps
6. Ensure the cache directory exists as described in chapter Special Directories.

7. If you want to sign documents, proceed with setting up your cryptographic provider as described in chapter Cryptographic Provider.
8. If you want to stamp text, proceed with setting the fonts required as described in chapter Fonts.

2.2.2 macOS

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

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

2.3 Interfaces

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

Interface	Programming Languages
`.NET`	The MS software platform .NET can be used with any .NET capable programming language such as: ■ C# ■ VB .NET ■ J# ■ others This interface is available in the Windows version only.
`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.
`PHP`	The PHP interface is available on Windows and Linux. Supported PHP versions are PHP 5.6 & 7.0 (Non Thread Safe).

2.3.1 Development

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

Req. This file is required for this interface.

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

Doc. This file is for documentation only.

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

Files for Development

Name	.NET	Java	COM	C	PHP
bin\‹platform›\PdfSecureAPI.dll	Req.	Req.	Req.	Req.	Req.
bin*NET.dll	Req.				
bin*NET.xml	Doc.				
bin\‹platform›\php*_pdftools.dll					Req.
doc*.pdf	Doc.	Doc.	Doc.	Doc.	Doc.
doc\PdfSecureAPI.idl			Doc.		
doc\javadoc*.*		Doc.			
doc\pdfsecure_doc.php and pdftoolsenums_doc.php					Doc.
include\pdfsecureapi_c.h				Req.	
include*.*				Opt.	
jar\SECA.jar		Req.			
lib\‹platform›\PdfSecureAPI.lib				Req.	
samples*.*	Doc.	Doc.	Doc.	Doc.	Doc.

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

File Description

Name	Description
bin\‹platform›\PdfSecureAPI.dll	This is the DLL that contains the main functionality (required).
bin*NET.dll	The .NET assemblies are required when using the .NET interface. The files bin*NET.xml contain the corresponding XML documentation for MS Visual Studio.
doc*.*	Various documentations.
include*.*	Contains files to include in your C / C++ project.

File Description

`lib\‹platform›\PdfSecureAPI.lib`	The object file library needs to be linked to the C/C++ project.
`jar\SECA.jar`	The Java API archive.
`bin\‹platform›\php*_pdftools.dll`	The `PdfTools` PHP extension must be added to the PHP extension directory.
`samples\*.*`	Contains sample programs in different programming languages.

2.3.2 Deployment

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

Files for Deployment

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

The deployment of an application works as described below:

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

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

1. The developed and compiled application consists of the file `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 PowerUser privileges and is added to the batch script.

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

```
regsvr32 /s %targetlocation%\PdfSecureAPI.dll.
```

2.4 Interface Specific Installation Steps

2.4.1 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 the registration of PdfSecureAPI.dll. Note that in Windows Vista and later, the command needs to be executed from an administrator shell.

```
regsvr32 "C:\Program Files\PDF Tools AG\bin\‹platform›\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`.[2]

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`.

2.4.2 Java Interface

The 3-Heights™ PDF Security API requires Java version 6 or higher.

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

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

For execution Additionally the library `PdfSecureAPI.dll` needs be in one of the system's library directories[3] or added to the Java system property `java.library.path`. This can be achieved by either adding it dynamically at program startup before using the API, or by specifying it using the switch `-Djava.library.path` when starting the Java VM. Choose the correct subdirectory `x64` or `Win32` depending on the platform of the Java VM[4].

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

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

[3] On Windows defined by the environment variable PATH and e.g. on Linux defined by LD_LIBRARY_PATH.

[4] 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`

```
-Djava.library.path=C:\Program Files\PDF Tools AG\bin\x64 sample
```

Note that on Unix-type systems, the path separator usually is a colon and hence the above changes to something like:

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

2.4.3 .NET Interface

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

The .NET assemblies (`*NET.dll`) are to be added as references to the project. They are required at compilation time.

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

The native DLL `PdfSecureAPI.dll` is called by the .NET assembly `PdfSecureNET.dll`.

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

Alternatively the directory where `PdfSecureAPI.dll` resides can be added to the environment variable `%Path%` or it can simply be copied manually to the output directory.

2.4.4 C Interface

- The header file `pdfsecureapi_c.h` needs to be included in the C/C++ program.
- 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%`).

2.4.5 PHP Interface

> **Note:** The descriptions below are valid for Unix-type systems. On a Windows system the libraries have a file extension `dll` instead of `so`.

The PHP interfaces for all 3-Heights™ products are contained in a sole "`PdfTools`" PHP extension. If multiple 3-Heights™ products are used, this extension is needed only once. Supported PHP versions are PHP 5.6 and 7.0 (both non thread-safe). The corresponding PHP extension libraries are `php56_pdftools.so` and `php70_pdftools.so`, henceforth summarized as `php‹xy›_pdftools.so`.

General Steps

- Copy the library `php‹xy›_pdftools.so` to the PHP extensions directory. This directory is configured in the PHP section of your PHP configuration file `php.ini` in the key `extension_dir`.
- Add the following line to the PHP section of your PHP configuration file `php.ini`:

```
extension=php‹xy›_pdftools.so
```

- Make sure that the native library `libPdfSecureAPI.so` is located in one of the system's library directories[3].

Additional Remarks for Command Line Use of PHP

- You can print out the current PHP configuration as configured in `php.ini` on the command line with:

```
php -i
```

- You can check whether PHP loads the `PdfTools` extension with:

```
php -m
```

Additional Remarks for Use in a Web Server

- You can locate the currently active PHP configuration file `php.ini` and report loaded modules with the following test PHP script:

```
<?php
phpinfo();
?>
```

- For an Apache web server to successfully locate and load the native library `libPdfSecureAPI.so` follow these steps:
  a. Create a file `/etc/ld.so.conf.d/pdf-tools.conf` that contains the full path to the directory where `libPdfSecureAPI.so` resides.
  b. Execute the command:

```
sudo ldconfig
```

  c. Restart the web server.
  d. Reload the test PHP script and check whether there is an entry for the `PdfTools` extension.

2.5 Uninstall, Install a New Version

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

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

2.6 Note about the Evaluation License

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

2.7 Special Directories

2.7.1 Directory for temporary files

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

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

Windows

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

Unix

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

2.7.2 Cache Directory

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

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

Windows

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

Linux, macOS and other Unixes

- 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.

2.7.3 Font Directories

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

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

Windows

1. `%SystemRoot%\Fonts`
2. directory `Fonts`, which must be a direct sub-directory of where `PdfSecureAPI.dll` resides.

macOS

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

Linux and other Unixes

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

3 License Management

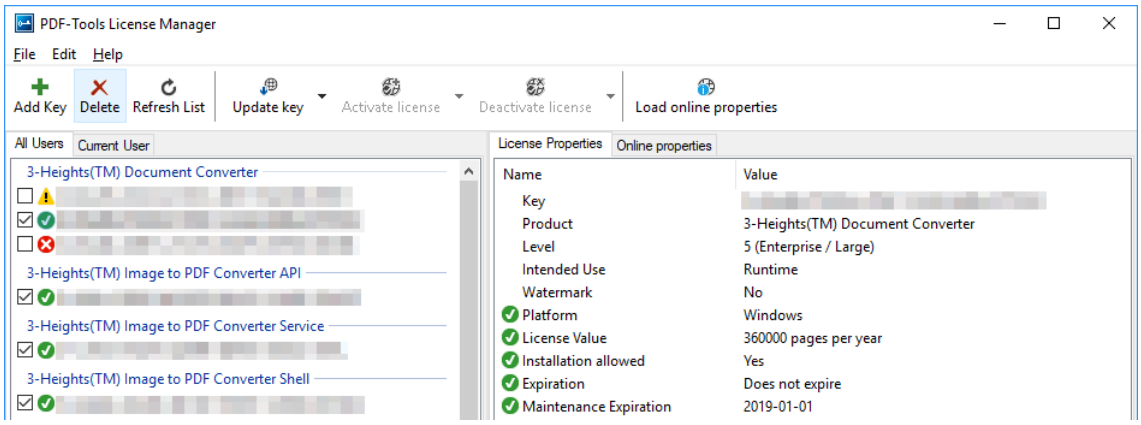
3.1 License Installation and Management

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

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

3.1.1 Graphical License Manager Tool

The GUI tool `LicenseManager.exe` is located in the `bin` directory of the product kit (Windows only).



List all installed license keys

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

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

Add and delete license keys

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

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

Display the properties of a license

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

3.1.2 Command Line License Manager Tool

The command line license manager tool `licmgr` is available in the `bin\x86` and `bin\x64` directory.

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

```
licmgr
```

List all installed license keys:

```
licmgr list
```

The currently active license for a specific product is marked with a star '*' on the left side.

Add and delete license keys:

Install new license key:

```
licmgr store 1-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

Delete old license key:

```
licmgr delete 1-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

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

g For all users

u Current user

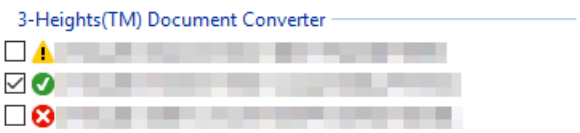
3.2 License Selection and Precedence

3.2.1 Selection

If multiple keys for the same product are installed in the same scope, only one of them can be active at the same time.

Installed keys that are not selected are not considered by the software!

In the Grahical User Interface use the check box on the left side of the license key to mark a license as selected.



With the Command Line Interface use the `select` subcommand:

```
licmgr select 1-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

3.2.2 Precedence

License keys are considered in the following order:

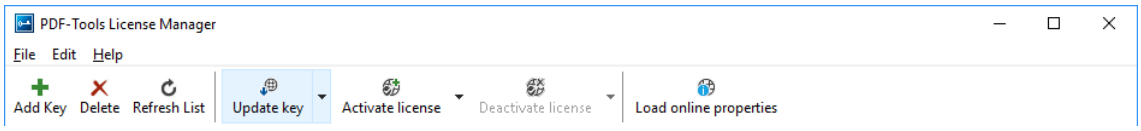
1. License key passed at runtime.
2. License selected for the current user
3. License selected for the current user (legacy key format)
4. License selected for all users
5. License selected for all users (legacy key format)

The first matching license is used, regardless whether it is valid or not.

3.3 Key Update

If a license property like the maintenance expiration date changes, the key can be update directly in the license manager.

In the Grahical User Interface select the license and press the button "Update Key" in the toolbar:



With the Command Line Interface use the `update` subcommand:

```
licmgr update 1-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

3.4 License activation

New licenses keys have to be activated (except for OEM licenses). These keys have to be installed in the license manager and may not be passed to the component at runtime.

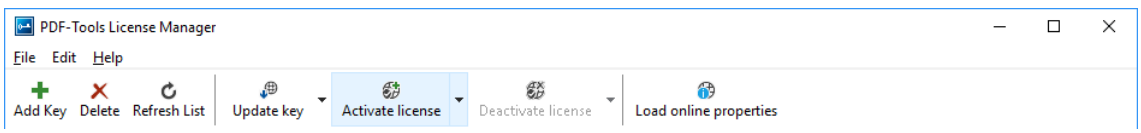
The license activation is tied to a specific computer. If the license is installed at user scope, the activation is also tied to that specific user. The same license key can be activated multiple times, if the license quantity is larger than 1.

Every license key includes a date, after which the license has to be activated, which is typically 10 days after the issuing date of the key. Prior to this date, the key can be used without activation and without any restrictions.

3.4.1 Activation

The License can be activated directly within the license manager. Every activation increases the activation count of the license by 1.

In the Grahical User Interface select the license and press the button "Activate license" in the toolbar:



With the Command Line Interface use the `activate` subcommand:

```
licmgr activate 1-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

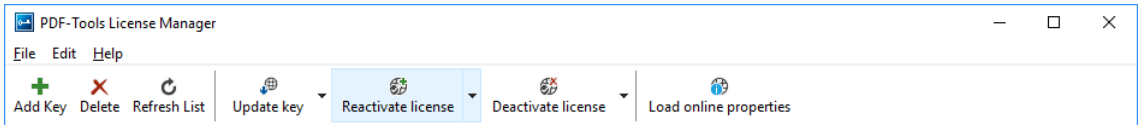
Note that the key has to be installed first.

3.4.2 Reactivation

The activation is tied to specific properties of the computer like the MAC address or host name. If one of these properties changes, the activation becomes invalid and the license has to be reactivated. A reactivation does **not** increase the activation count on the license.

The process for reactivation is the same as for the activation.

In the Grahical User Interface the button "Activate license" changes to "Reactivate license":



With the Command Line Interface the subcommand `reactivate` is used instead of `activate`:

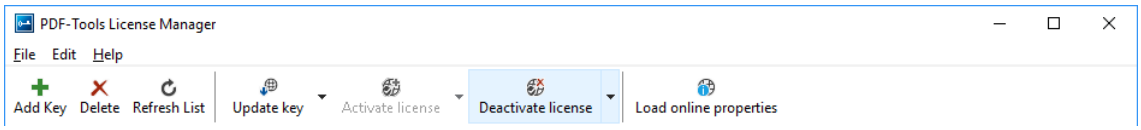
```
licmgr reactivate 1-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

3.4.3 Deactivation

To move a license to a different computer, it has to be deactivated first. Deactivation decreases the activation count of the license by 1.

The process for deactivation is similar to the activation process.

In the Grahical User Interface select the license and press the button "Deactivate license" in the toolbar:



With the Command Line Interface use the `deactivate` subcommand:

```
licmgr deactivate 1-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

3.5 Offline Usage

The following actions in the license manager need access to the internet:

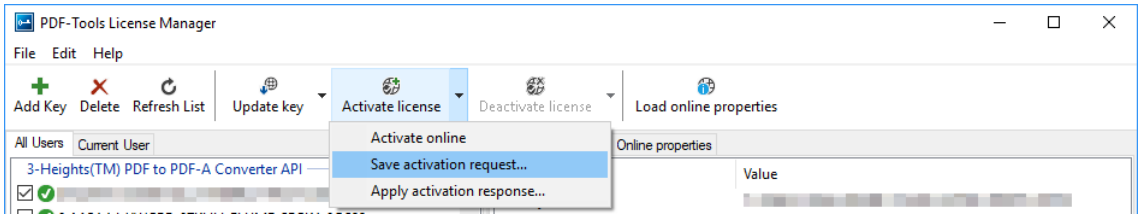
- License Activation
- License Reactivation

- License Deactivation
- Key Update

On systems wihout internet access, a three step process can be used instead, using a form on the PDF Tools website.

3.5.1 First Step: Create a Request File

In the Grahical User Interface select the license and use the dropdown menu on the right side of the button in the toolbar:



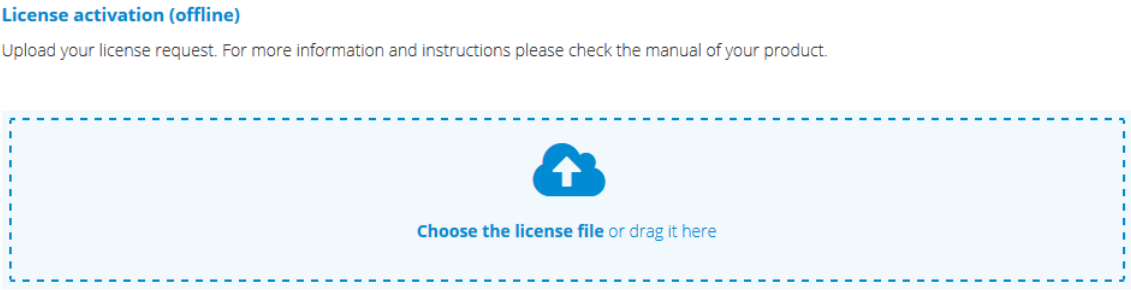
With the Command Line Interface use the `-fs` option to specify the destination path of the request file:

```
licmgr activate -fs activation_request.bin 1-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

> **License Deactivation:** When saving the deactivation request file, the license is **deactivated immediately** and cannot be used any further. It can however only be activated again after completing the deactivation on the website.

3.5.2 Second Step: Use Form on Website

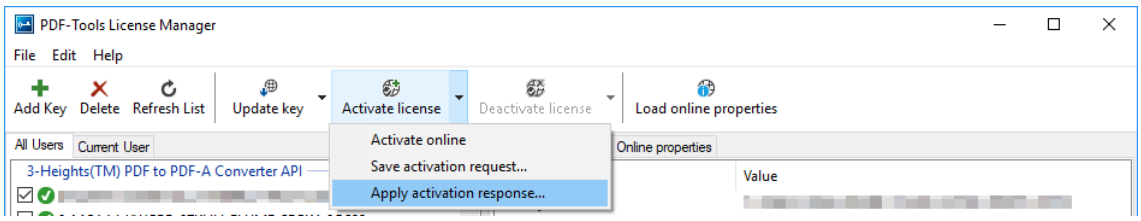
Open the following website in a web browser: http://www.pdf-tools.com/pdf20/en/mypdftools/licenses-kits/license-activation/ Upload the request by dragging it onto the marked area:



Upon success, the response will be downloaded automatically if necessary.

3.5.3 Third Step: Apply the Response File

In the Grahical User Interface select the license and use the dropdown menu on right side of the button in the toolbar:



With the Command Line Interface use the `-fl` option to specify the source path of the response file:

```
licmgr activate -fl activation_response.bin 1-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

3.6 License Key Versions

As of 2018 all new keys will have the format `1-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX`. Legacy keys with the old format `0-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX` are still accepted for a limited time period.

For compatibility reasons, old and new version keys can be installed side by side and one key of each version can be selected at the same time. In that case, the software always uses the new version.

3.7 License Key Storage

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

3.7.1 Windows

The license keys are stored in the registry:

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

3.7.2 macOS

The license keys are stored in the file system:

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

3.7.3 Unix/Linux

The license keys are stored in the file system:

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

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

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

3.8 Troubleshooting

3.8.1 License key cannot be installed

The license key cannot be installed in the license manager application. The error message is: `"Invalid license format."`

Possible causes:

- The license manager application is an older version that only supports the legacy key format.

Solution

Use a current version of the license manager application or use a license key in the legacy key format if available.

3.8.2 License is not visible in license manager

The license key was successfully installed previously but is not visible in the license manager anymore. The software is still working correctly.

Possible causes:

- The license manager application is an older version that only supports the legacy key format.

Solution

Use a current version of the license manager application.

3.8.3 License is not found at runtime

The license is not found at runtime by the software. The error message is: `"No license key was set."`

Possible causes:

- The license key is actually missing (not installed).
- The license key is installed but not selected in the license manager.
- The application is an older version that only supports the legacy key format, while the license key has the new license format.

Solution

Install and select a valid license key that is compatible with the installed version of the software or use a newer version of the software. The new license key format is supported starting with version 4.10.26.1

For compatibility reasons, one license key of each format can be selected at the same time.

3.8.4 Eval watermark is displayed where it should not

The software prints an evaluation watermark onto the output document, even if the installed license is a productive one.

Possible causes:

- There is an evaluation license key selected for the **current user**, that takes precedence over the key for **all users**.

> **Note:** The software might be run under a different user than the license manager application.

- There is an evaluation license key selected with a newer license format that takes precedence over the key in the older format.
- The software was not restarted after changing the license key from an evaluation key to a productive one.

Solution

Disable or remove all evaluation license in all scopes and restart the software.

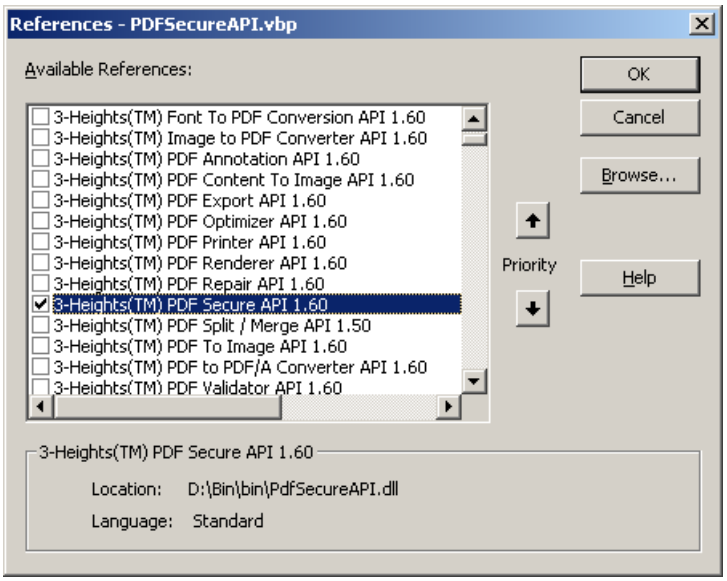
4 Programming Interfaces

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, here is a quick start guide for you:

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



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

```
Private Sub Command1_Click()
   Dim 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.

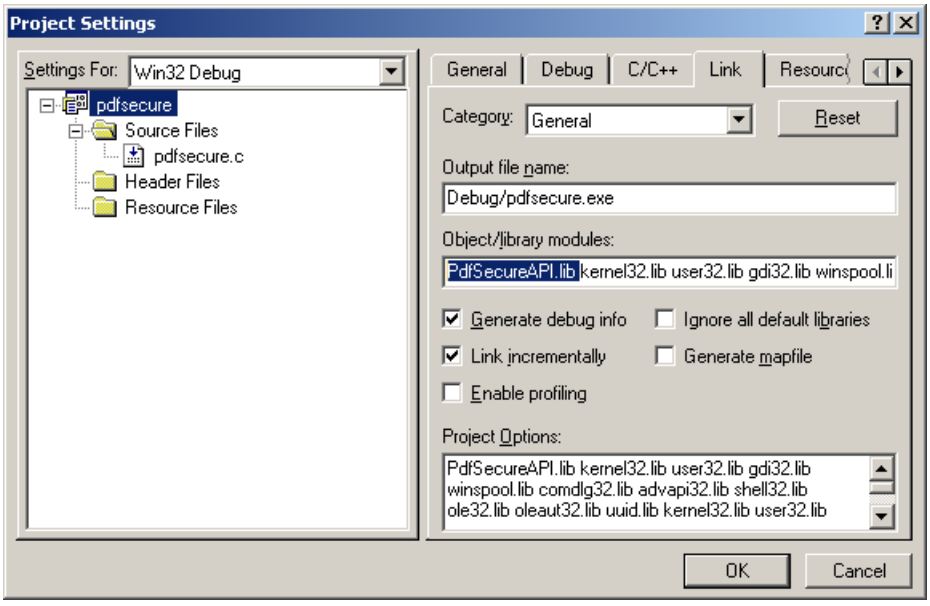
```
Private Sub CreateOutput_Click()
   Dim doc As New PDFSECUREAPILib.PdfSecure
```

```
    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
```

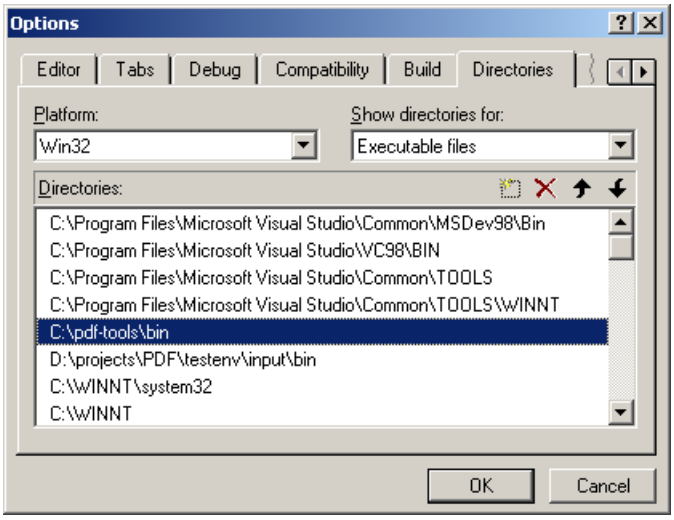
4.2 C/C++

In order 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`)



3. Add the path where the dynamic link library `PdfSecureAPI.dll` resides to the "Executable files directories". E.g. as shown in the screenshot below. In most cases it suffices to simply add it to the environment variable `Path`.



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.

```
#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;
}
```

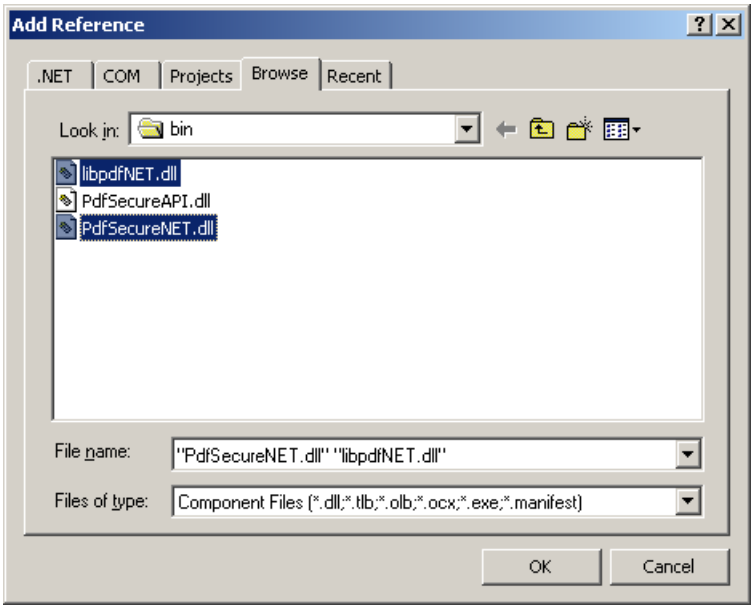
4.3 .NET

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 for a quick start is to refer to this sample.

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

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

To do so, in the “Solution Explorer” right-click your project and select “Add Reference…”. The “Add Reference” dialog will appear. In the tab “Browse”, browse for the .NET assemblies `libpdfNET.dll` and `PdfSecureNET.dll`.
Add them to the project as shown below:

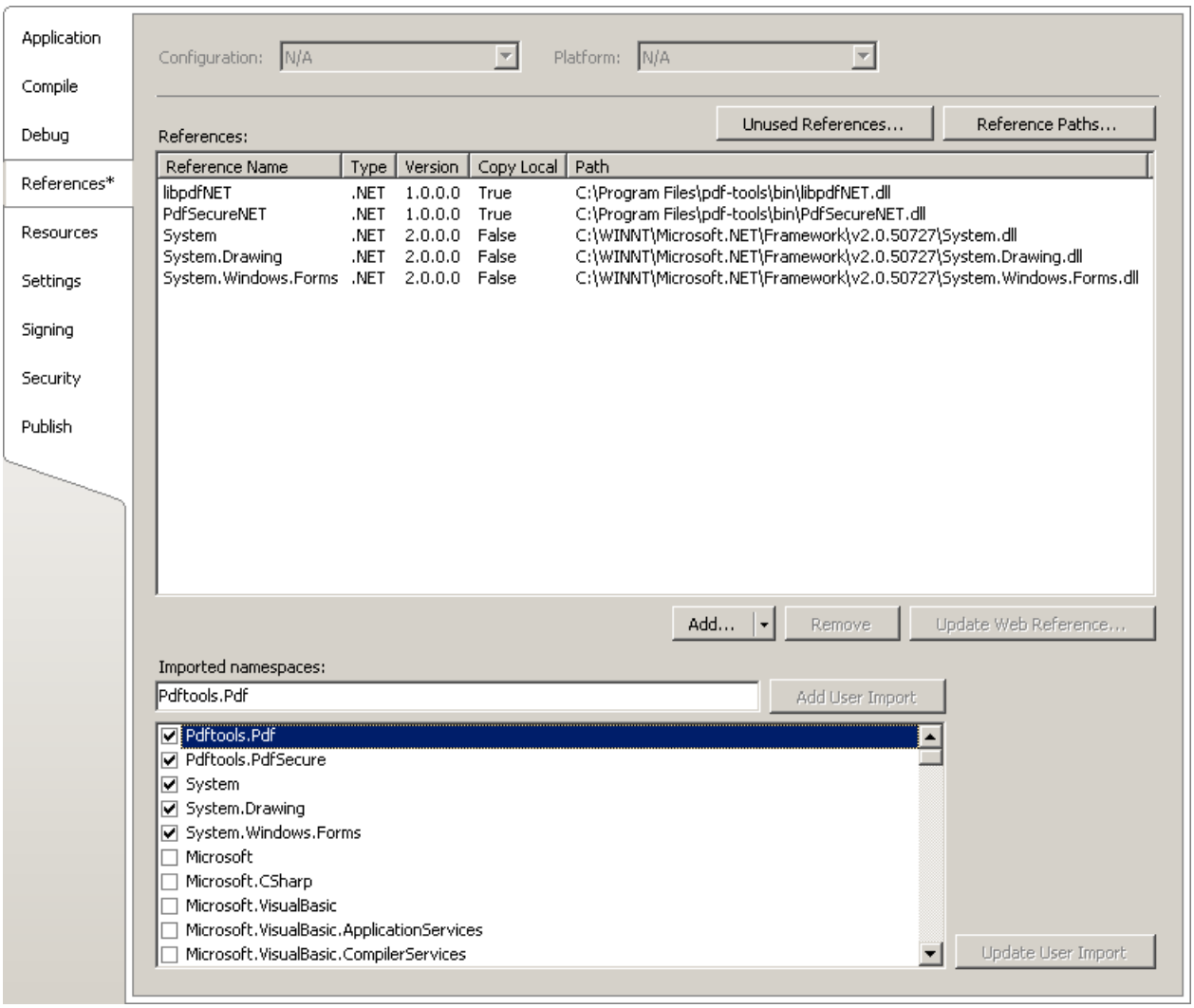


3. Import namespaces (Note: This step is optional, but useful.)
4. Write your code.

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

4.3.1 Visual Basic

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



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. Hereby it is crucial that depending on the solution platform (x86 or x64) the matching native DLL `PdfSecureAPI.dll` (from the directory `bin\Win32` or `bin\x64`) is copied to the output directory.
2. The executable is created in the directory `bin\Release`.
3. For deployment, the executable and all .NET assemblies must be copied into the same folder on the target computer. The .NET assemblies of the 3-Heights™ PDF Security API have the file name `bin\*NET.dll`.
4. At runtime, the native DLL `PdfSecureAPI.dll` must be found on the target computer by the DLL search sequence. To ensure this, the DLL must either be copied to the folder containing the executable or to a directory on the environment variable `Path` (e.g. `%SystemRoot%\system32`).
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, distinguishable 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 the respective issue.

Troubleshooting: DllNotFoundException

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

Resolve this by either:

- adding `PdfSecureAPI.dll` as an existing item to your project and set its property "Copy to output directory" to "Copy if newer", or
- adding the directory where `PdfSecureAPI.dll` resides to the environment variable `%Path%`, or
- copying `PdfSecureAPI.dll` to the output directory of your project.

Troubleshooting: BadImageFormatException

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

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

AnyCPU This means, that the application will run as a 32-bit process on 32-bit Windows and as 64-bit process on 64-bit Windows. When using AnyCPU one has to use a different native DLL, depending on the platform of Windows. This can be ensured either when installing the application (by installing the matching native DLL) or at application start-up (by determining the application's platform and ensuring the matching native DLL is loaded).

x86 This means, that the application will always run as 32-bit process, regardless of the platform of the Windows installation. The 32-bit DLL runs on all systems, which makes this the simplest configuration. Hence, if an application needs to be portable and does not require any specific 64-bit features, it is recommended to use this setting.

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

5 User's Guide

5.1 Overview of the API

5.1.1 What is the 3-Heights™ PDF Security API about?

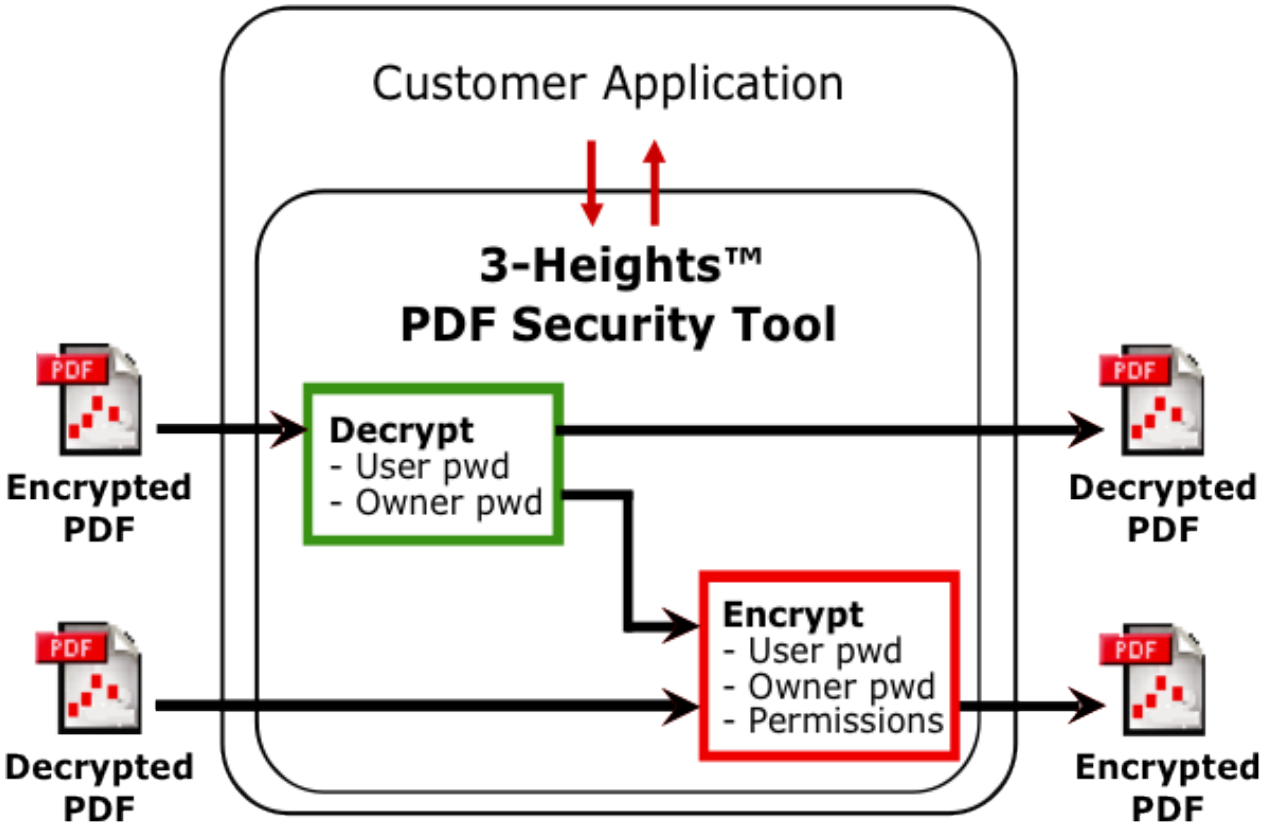
The 3-Heights™ PDF Security API provides three key functionalities related to security in PDF documents:

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. What encryption and what a digital signature is, is described in the upcoming chapters.

5.2 How does the API work in general?

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 which, 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

What operations in a PDF document are granted is controlled via its permission flags. In order 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 form and digitally signing the document
- Assembling the document (inserting, rotating, deleting pages, etc.)

5.3.4 How to Encrypt 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. Note that 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 such as 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 How to Read an Encrypted PDF Document

A PDF document which 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 that looks like this:

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

A PDF document which 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 will 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 in order to be able to display them. It technically 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 which 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 he provided. Any PDF application which 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. E.g. a brute force attempt testing all passwords which 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.

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, their private keys and implements cryptographic algorithms.

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

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

On Windows, the certificate is usually also available in the Windows Cryptographic Provider.

If you need to sign documents on a non-Windows system with an USB token that does not come with middleware for your platform, you can use the 3-Heights™ Signature Creation and Validation Service.

If you need to sign documents on Windows in a non-interactive or locked session[5], use the 3-Heights™ Signature Creation and Validation Service.

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

For more information and installation instructions consult the separate document `TechNotePKCS11.pdf`.

[5] See the description of the 3-Heights™ Signature Creation and Validation Service for more details on this topic.

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
  - PKCS#11 softtoken on Solaris

  For more information and installation instructions of the above stores consult the separate document `TechNotePKCS11.pdf`.
- *Windows*: If no PKCS#11 provider is available, soft certificates can be imported into Windows certificate store, which can then be used as cryptographic provider →Windows Cryptographic Provider

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

- 3-Heights™ Signature Creation and Validation Service
- SwissSign Digital Signing Service
- SwissSign SuisseID Signing Service
- QuoVadis sealsign
- Swisscom All-in Signing Service

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

- The SuisseID USB Tokens use `cvP11.dll`
- The CardOS API from Atos (Siemens) uses `siecap11.dll`
- The IBM 4758 cryptographic coprocessor uses `cryptoki.dll`
- Devices from Aladdin Ltd. use `etpkcs11.dll`

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

`‹Pin›` is optional, if it is not defined, the submission for the pin is activated via the pad of the token.

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

Example:

```
Provider = "C:\Windows\system32\siecap11.dll;4;123456"
```

> **Note:** Some PKCS#11 drivers require the `Terminate` method to be called. Otherwise your application might crash upon termination.

The chapter Guidelines for Mass Signing contains important information to optimize performance when signing multiple documents.

Interoperability Support

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

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

Selecting a Certificate for Signing

The 3-Heights™ PDF 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`
   - SHA1 fingerprint of the certificate. The fingerprint is 20 bytes long and can be specified in hexadecimal string representation, e.g. "b5 e4 5c 98 5a 7e 05 ff f4 c6 a3 45 13 48 0b c6 9d e4 5d f5". In Windows certificate store this is called "Thumbprint", if "Thumbprint algorithm" is "sha1".
2. **Certificate Issuer and SerialNumber**
   Properties `Issuer` and `SerialNumber`
   - Certificate Issuer (e.g. "QV Schweiz CA"), in Windows certificate store this is called "Issued By".
   - Serial number of the certificate (hexadecimal string representation, e.g. "4c 05 58 fb"). This is a unique number assigned to the certificate by its issuer. In Windows certificate store this is the field called "Serial number" in the certificate's "Details" tab.
3. **Certificate Name and optionally Issuer**
   Properties `Name` and `Issuer`
   - Common Name of the certificate (e.g. "PDF Tools AG"), in Windows certificate store this is called "Issued To".
   - Optional: Certificate Issuer (e.g. "QV Schweiz CA"), in Windows certificate store this is called "Issued By".

Using PKCS#11 stores with missing issuer certificates

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

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

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

Cryptographic Suites

Message Digest Algorithm

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

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

SHA-256 (default)

SHA-384

SHA-512

RIPEMD-160

Signing Algorithm

The signing algorithm can be configured by setting the provider session property `SigAlgo`. Supported values are:

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

RSA_SSA_PSS This algorithm is sometimes also called RSA-PSS.

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

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

5.5.2 Windows Cryptographic Provider

This provider uses Windows infrastructure to access certificates and to supply cryptographic algorithms. Microsoft Windows offers two different APIs, the Microsoft CryptoAPI and Cryptography API Next Generation (CNG).

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

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

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

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

CNG Support of SuisseID: SuisseID supports CNG starting with middleware version 3.6.2. When using an older middleware version, an upgrade is highly recommended.

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. Note that the use of SHA-1 is strongly discouraged by the cryptographic community.

Configuration

Provider Property `Provider` or argument of `BeginSession`

The provider configuration string has the following syntax:

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

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

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

If `‹Provider›` is omitted, the default provider is used. The default provider is suitable for all systems where CNG is available.

Examples: Use the default provider with no pin.

```
Provider = ""
```

Examples: "123456" being the pin code.

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

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

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

Certificate Store Property Store

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

Store Location Property StoreLocation

Either of the following store locations

- “Local Machine”
- “Current User” (default)

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

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

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

Selecting a Certificate for Signing

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

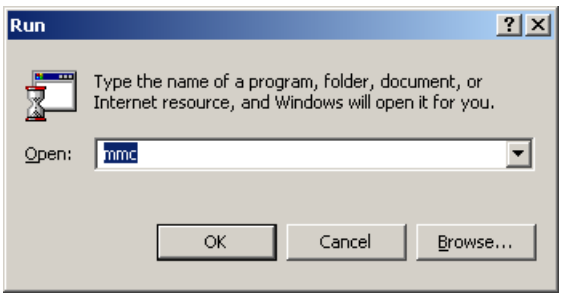
Certificates

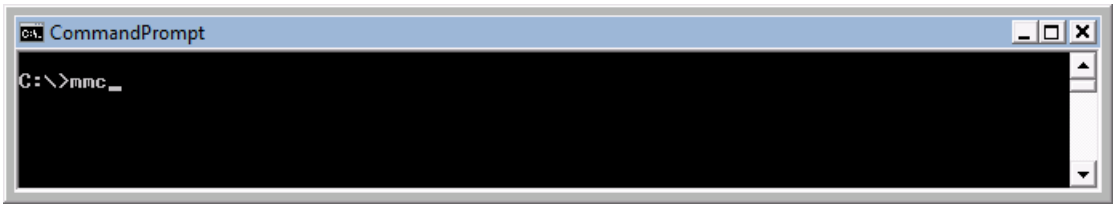
In order to sign a PDF document, a valid, existing certificate name must be provided and its private key must be available.

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

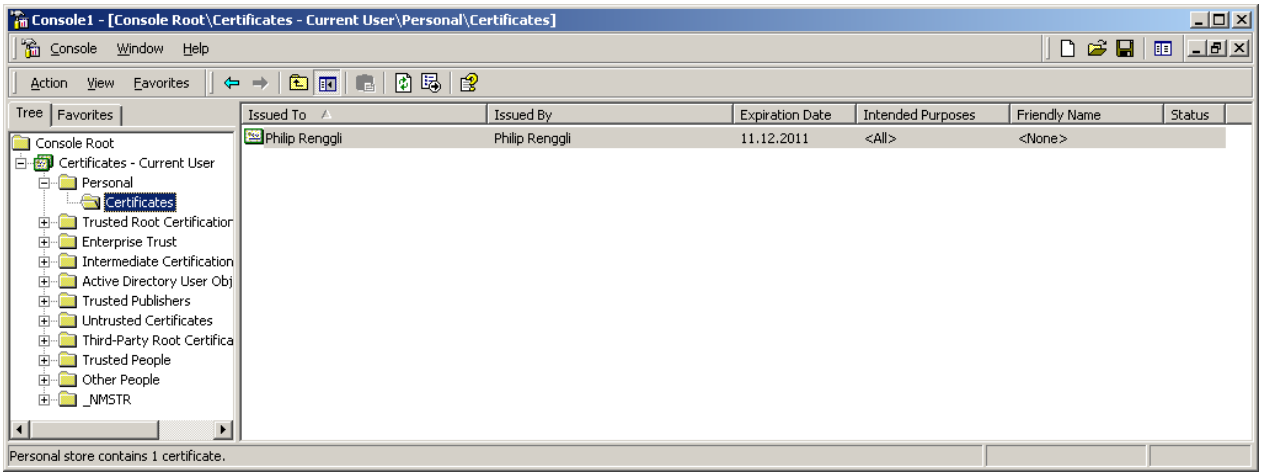
On the Windows operating system certificates can be listed by the Microsoft Management Console (MMC), which is provided by Windows. In order to see the certificates available on the system, do the following steps:

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

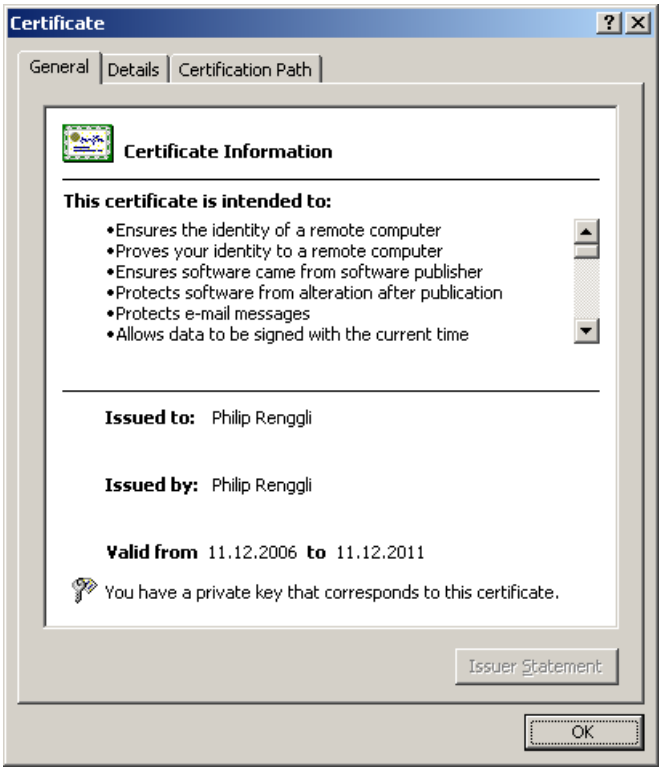




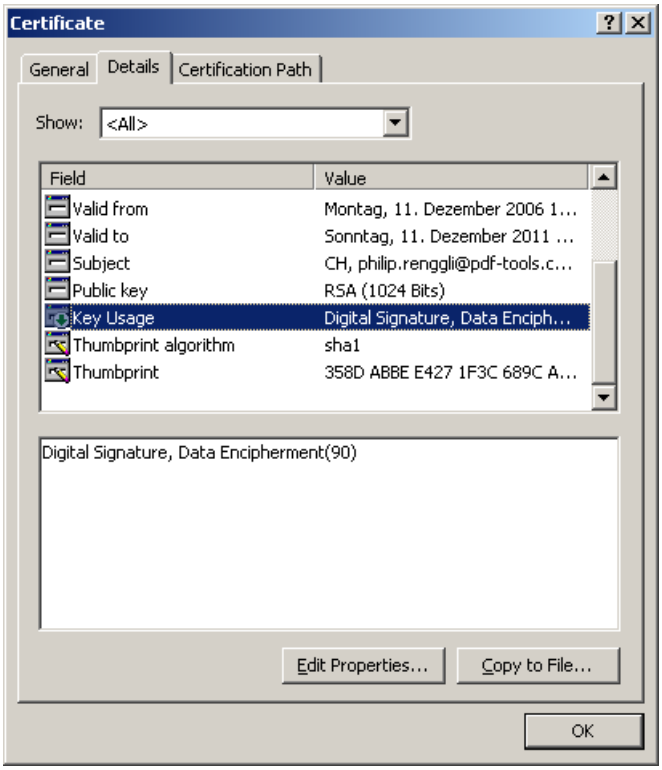
2. Under “File” →“Add/Remove Snap-in”
3. Choose “Certificates” and click the “Add” button
4. In the next window choose to manage certificates for “My user account”
5. Click “Finish”
6. The certificate must be listed under the root “Certificates - Current User”, for example as shown in the screenshot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



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

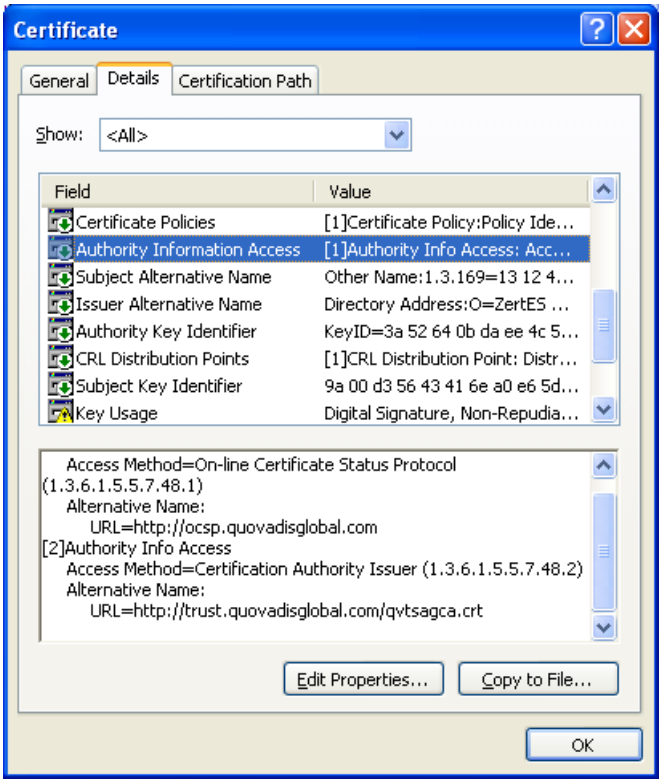


8. In the tab Detail of the certificate, there is a field named “Key Usage”. This field must contain the value “Digital Signature”. Additional values are optional, see also screenshot.
   You must have the private key that corresponds to this certificate.



Qualified Certificates

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



Cryptographic Suites

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

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

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

5.5.3 3-Heights™ Signature Creation and Validation Service

The 3-Heights™ Signature Creation and Validation Service provides HTTP protocol based remote access to cryptographic providers such as smartcards, USB tokens, and other cryptographic infrastructure such as HSMs.

Use of the 3-Heights™ Signature Creation and Validation Service provides the following advantages:

1. By means of this service the tokens can be hosted centrally and used by any client computer which has access to the service.
2. Cryptographic devices that can be used on Windows only can be made accessible to siging processes running on Non-Windows systems.
3. Cryptographic devices can be made accessible to processes running in non-interactive sessions. Many cryptographic devices must always be used in an interactive session for two reasons.
   First, the middleware requires the user to enter the pin interactively to create a qualified electronic signature.

Second, USB tokens and smart cards are managed by Windows such that the device is available only to the user currently using the computer's console. Therefore, services, remotely logged in users and applications running in locked sessions have no access to the device.

> **Note:** that this is a separate product and this chapter describes its usage with the 3-Heights™ PDF Security API only.

For more information on the 3-Heights™ Signature Creation and Validation Service and installation instructions, please refer to its separate user manual.

Configuration

Provider Property `Provider` or argument of `BeginSession`

The provider configuration string has the following syntax:

`"http://server.mydomain.com:‹port›/‹token›;‹password›"`

Where:

- `server.mydomain.com` is the hostname of the server
- `‹port›` is optional, port of the server.
- `‹token›` the ID of the token.
- `‹password›` password of the token.

Example:

```
Provider = "http://server.mydomain.com:8080/0001;pass01"
```

A more detailed description can be found in the user manual of the 3-Heights™ Signature Creation and Validation Service.

Selecting a Certificate for Signing

Selection of the signing certificate works the same as if the token was used directly: Selecting a Certificate for Signing.

Cryptographic Suites

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

The `MessageDigestAlgorithm` must be set by the client.

The `SigAlgo` must be configured in the server's token configuration file `TokenConfig.xml`.

5.5.4 SwissSign Digital Signing Service

Provider Property `Provider` or argument of `BeginSession`

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

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

There are two types of properties:

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

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

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

Description	Required	Value	Setting
Common Name	required	The name of the signer must be set[6].	Property `Name`.

Time-stamp	optional	Use the value `urn:ietf:rfc:3161` to embed a time-stamp.	Property `TimeStampURL`
Revocation Info	recommended	To embed OCSP responses or CRL.	Property `EmbedRevocationInfo`
Visual Appearance	optional	See separate chapter How to Create a Visual Appearance of a Signature.	

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

5.5.5 SwissSign SuisseID Signing Service

In order to use the SuisseID Signing Service, please contact Swiss Post Solutions AG (`suisseid@post.ch`) to obtain access credentials. Prior to invoking the SuisseID Signing Service, user authentication via the SuisseID Identity Provider (IDP) is a pre-requisite. So the calling application must integrate via SAML (e.g. SuisseID SDK) with the SuisseID Identity Provider. The IDP issues SAML tokens upon successful user authentication.

> **Note:** The name of the signature should be the signer's name (e.g. "`‹given-name› ‹surname›`"). The signer's name can be retrieved for the SAML token as the IDP provides this as qualified attributes (yellowid verified).

Provider Property `Provider` or argument of `BeginSession`

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

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

There are two types of properties:

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

Name	Type	Required	Value

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

SAMLToken	File	required	SAML token issued by the SuisseID Identity Provider (IDP). Example: `C:\temp\my-saml.xml` Note: The SAML token received from the IDP is a signed XML. It must be treated as binary data and not be modified in any way. For example, the token should not be read into a string or XML object and re-serialized.
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.

Description	Required	Value	Setting
Common Name	required	The name of the signer must be set[7].	Property `Name`.
Time-stamp	recommended	Use the value `http://tsa.swisssign.net` to embed a time-stamp.	Property `TimeStampURL`
Revocation Info	recommended	To embed OCSP responses or CRL.	Property `EmbedRevocationInfo`
Visual Appearance	optional	See separate chapter How to Create a Visual Appearance of a Signature.	

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

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

5.5.6 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.

Description	Required	Value	Setting
Common Name	required	The name of the signer must be set[8].	Property `Name`.
Time-stamp	-	Not available.	

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

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

5.5.7 Swisscom All-in Signing Service

General Properties

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

Description	Required	Value	Setting
Common Name	required	Name of the signer[9].	Property `Name`
Provider	required	The service endpoint URL of the REST service. Example: `https://ais.swisscom.com/AIS-Server/rs/v1.0/sign`	Property `Provider`
Time-stamp	optional	Use the value `urn:ietf:rfc:3161` to embed a time-stamp.	Property `TimeStampURL`
Revocation Info	optional	To embed OCSP responses	Property `EmbedRevocationInfo`

If a proxy is used for the connection to the service, see chapter How to Use a Proxy for more information.

Provider Session Properties

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

There are two types of properties:

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

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

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`

On-Demand Certificates

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

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

Step-Up Authorization using Mobile-ID

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

Name	Type	Required	Value
SwisscomAllInMSISDN	String	required	Mobile phone number. Example: `+41798765432`

SwisscomAllInMessage	String	required	The message to be displayed on the mobile phone. Example: `Pipapo halolu`.
SwisscomAllInLanguage	String	required	The language of the message. Example: `DE`

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

5.5.8 GlobalSign Digital Signing Service

Provider Property `Provider` or argument of `BeginSession`

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

- Productive: `https://emea.api.dss.globalsign.com:8443/v1.0`
- Test: `https://stg-emea.api.hvca.globalsign.com:8443` (IP `185.140.80.190`)

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[10].	Property `Name`.
Time-stamp	recommended	Use the value `urn:ietf:rfc:3161` to embed a time-stamp.	Property `TimeStampURL`
Revocation Info	recommended	To embed OCSP responses or CRL.	Property `EmbedRevocationInfo`
Visual Appearance	optional	See separate chapter How to Create a Visual Appearance of a Signature.	

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

How to create the SSL client certificate

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

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

How to get the SSL server certificate

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

1. Get the server's SSL certificate:

```
openssl s_client -showcerts -connect emea.api.dss.globalsign.com:8443 ^
```

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

```
  -cert clientcert.crt -key privateKey.key
```

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

Advice on using the service

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

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

Due to the short-lived nature of the signing certificates, it is important to embed revocation information immediately. For example by using `AddValidationInformation` or `EmbedRevocationInfo`. Furthermore it is highly recommended to embed a time-stamp in order to prove that the signature was created during the certificate's validity period.

5.5.9 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.

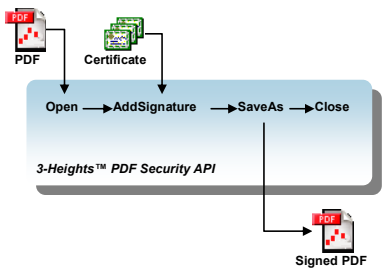
This allows for treating the PDF and signature technologies separately and also provides an easy way to replace a signature handler.

5.6 How to Create Digital Signatures

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

5.6.1 How to Sign a PDF Document

As we saw in the chapter How to Create Electronic Signatures, the process steps to add a signature are as shown in the graphic below:

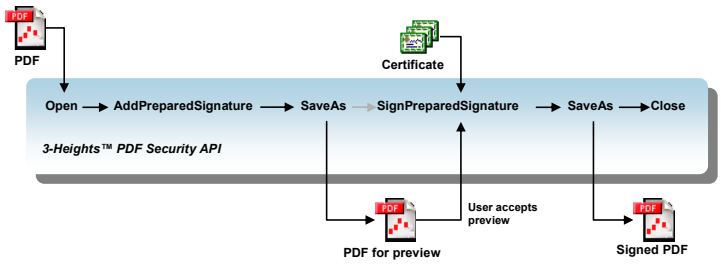


1. A PDF input-document is opened
2. A signature is created and added using a certificate

3. A new, signed PDF output-document is created
4. The input-document is closed

5.6.2 How to Create a Preview of a Signed Document

The 3-Heights™ PDF Security API provides the possibility to 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 change 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.

5.6.3 How to Create a PAdES Signature

The PAdES European Norm recommends to use one of the following four baseline signature levels.

PAdES-B-B A digital signature.

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

PAdES-B-LT A digital signature with a time-stamp token and signature validation data.

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

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

Requirements

For general requirements and preparation steps see chapter How to Create Electronic Signatures.

Requirements

Level	Signing Certificate	Time-stamp	Product
B-B	any	no	3-Heights™ PDF Security API
B-T	any	required	3-Heights™ PDF Security API
B-LT	advanced or qualified certificate	required	3-Heights™ PDF Security API
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 time-stamp signature, if your TSA server does not include them in the time-stamp.

A proper error handling is crucial in order to ensure the creation of correctly signed documents. The output document was signed successfully, if and only if the method `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 nor can encryption parameters be changed. When creating signature levels PAdES-B-B or PAdES-B-T that might 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 compliance is not sufficient.

Create a PAdES-B-B Signature

Input Document Any PDF document.

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

```
using (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";
    doc.AddSignature(sig);
```

```
  }

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

Create a PAdES-B-T Signature

Input Document Any PDF document.

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

```
using (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";
    signature.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);
}
```

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);
}
```

Create a PAdES-B-LTA Signature or Enlarge 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 enlarged.

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);
}
```

5.6.4 How to Apply 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.

Note that signing a linearized file renders the linearization information unusable. Therefore it is recommended to not linearize files that will be signed multiple times.

```
Dim Document As New PDFSECUREAPILib.PdfSecure
```

```
Document.Open "input.pdf", ""
Dim Signature1 As New PDFSECUREAPILib.PdfSignature
Signature1.Name = "First Signer"
Signature1.Provider = "cvp11.dll;0;secret-pin1"
Document.AddSignature Signature1
Document.SaveAs "tmp.pdf"
Document.Close

Document.Open "tmp.pdf", ""
Dim Signature2 As New PDFSECUREAPILib.PdfSignature
Signature2.Name = "Second Signer"
Signature2.Provider = "cvp11.dll;1;secret-pin2"
Document.AddSignature Signature2
Document.SaveAs "output.pdf"
Document.Close
```

5.6.5 How to Create a Time-stamp Signature

For a time-stamp signature no local signing certificate is required. Instead the time-stamp signature requested from the time-stamp Authority (TSA) is embedded into the document.

Example: Create a time-stamp 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);
}
```

A Cryptographic Provider is required on non-Windows systems only.

5.6.6 How to Create a Visual Appearance of a Signature

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

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

Different properties of the visual appearance can be specified.

Page and Position See properties `PageNo` and `Rect`.

Color See properties `FillColor` and `StrokeColor`.

Line Width The line width of the background rectangle, see property `LineWidth`.

Text Two text fragments can be set using two different fonts and font sizes, see properties `Text1`, `Text2`, `FontName1`, `FontName2`, `FontSize1`, and `FontSize2`.

Background image See property `ImageFileName`.

5.6.7 Guidelines for Mass Signing

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

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

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

1. Create a `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.

Signing concurrently using multiple threads

The 3-Heights™ PDF Security API is thread-safe. Each `PdfSecure` object should be used in one thread at the 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.

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");
```

5.6.8 Miscellaneous

Caching of CRLs, OCSP, and Time-stamp Reponses

In order 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 (Time-stamp Protocol) and data from signature services. This data is usually valid over period of time that is defined by the protocol, which is normally at least 24 hours. Caching improves the speed, because there are situations when the server does not need to be contacted for every digital signature.

The following caches are stored automatically by the 3-Heights™ PDF Security API at the indicated locations within the Cache Directory:

OCSP responses	`<CacheDirectory>/OCSP Responses/server-hash.der`
CRL	`<CacheDirectory>/CLRs/server.der`
Time stamp responses[11]	`<CacheDirectory>/Time Stamps/server.der`
Service data	`<CacheDirectory>/Signature Sizes/hash.bin`

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

How to Use a Proxy

The 3-Heights™ PDF 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"

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

Configuration of Proxy Server and Firewall

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

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

OCSP

- `application/ocsp-request`
- `application/ocsp-response`

Time-stamp

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

Signature services

- Signature service specific MIME types.

Setting the Signature Build Properties

In the signature build properties dictionary the name of the application that created the signature can be set using the provider session properties `Prop_Build.App.Name` and `Prop_Build.App.REx`. The default values are "3-Heights™ PDF Security API" and its version.

5.7 How to Validate Digital Signatures

5.7.1 Validation of a Qualified Electronic Signature

There are basically three items that need to be validated:

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

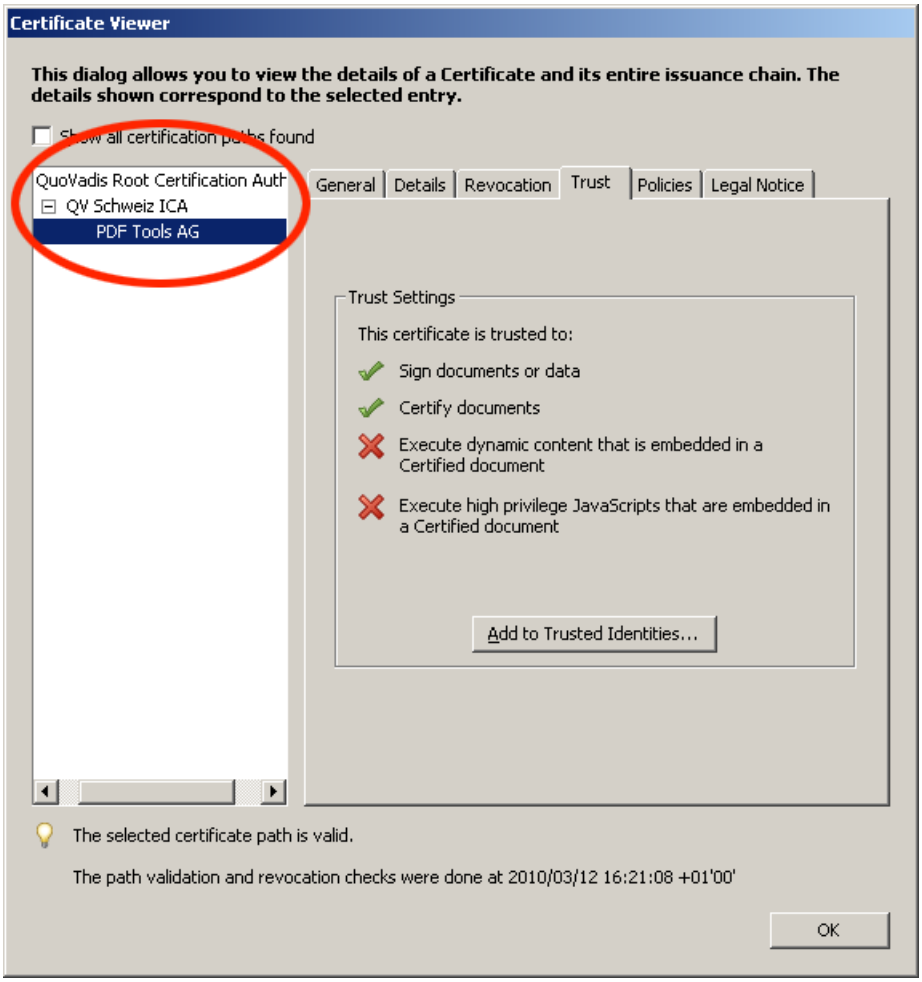
Validation can be in different ways, e.g. Adobe Acrobat, from which the screenshots below are taken.

Trust Chain

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

1. Retrieve a copy of the certificate containing a public key. This can be done 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.

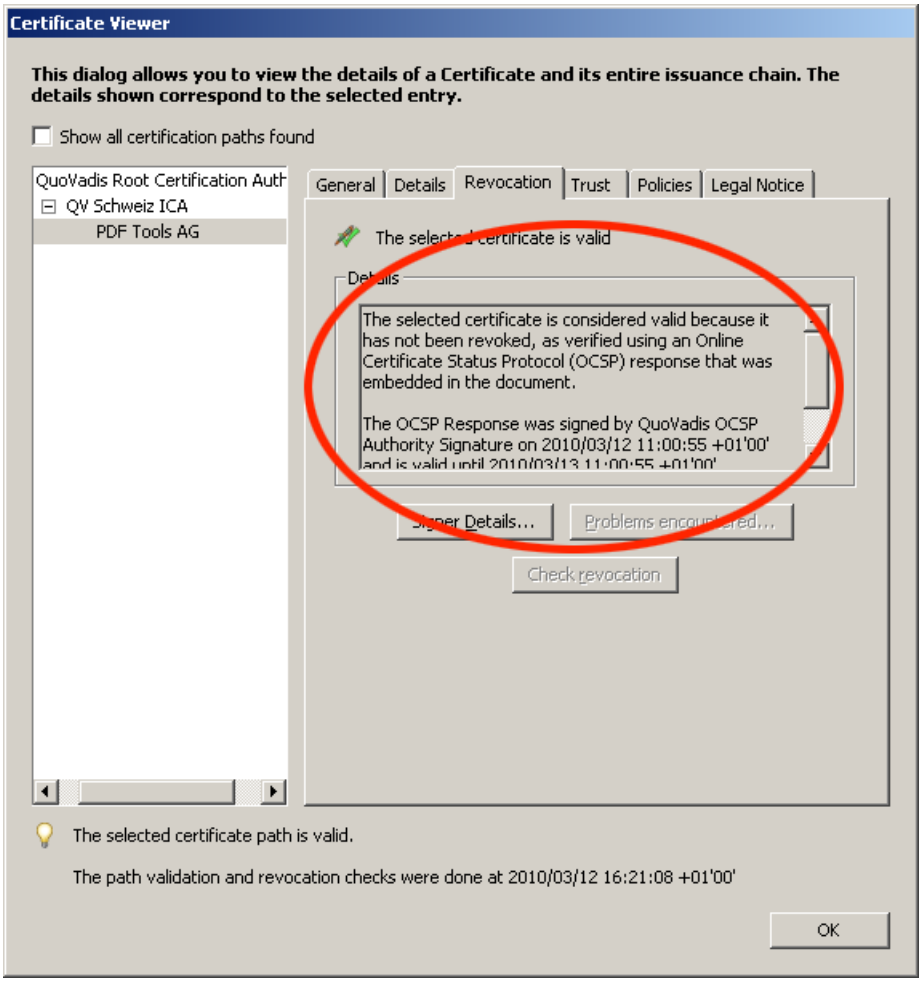
After that you can validate the signature, e.g. by open the PDF document in Adobe Acrobat, right-click the signature and select "Validate", then select "Properties" and select the tab "Trust". There the certificate should be trusted to "sign documents or data".



Revocation Information

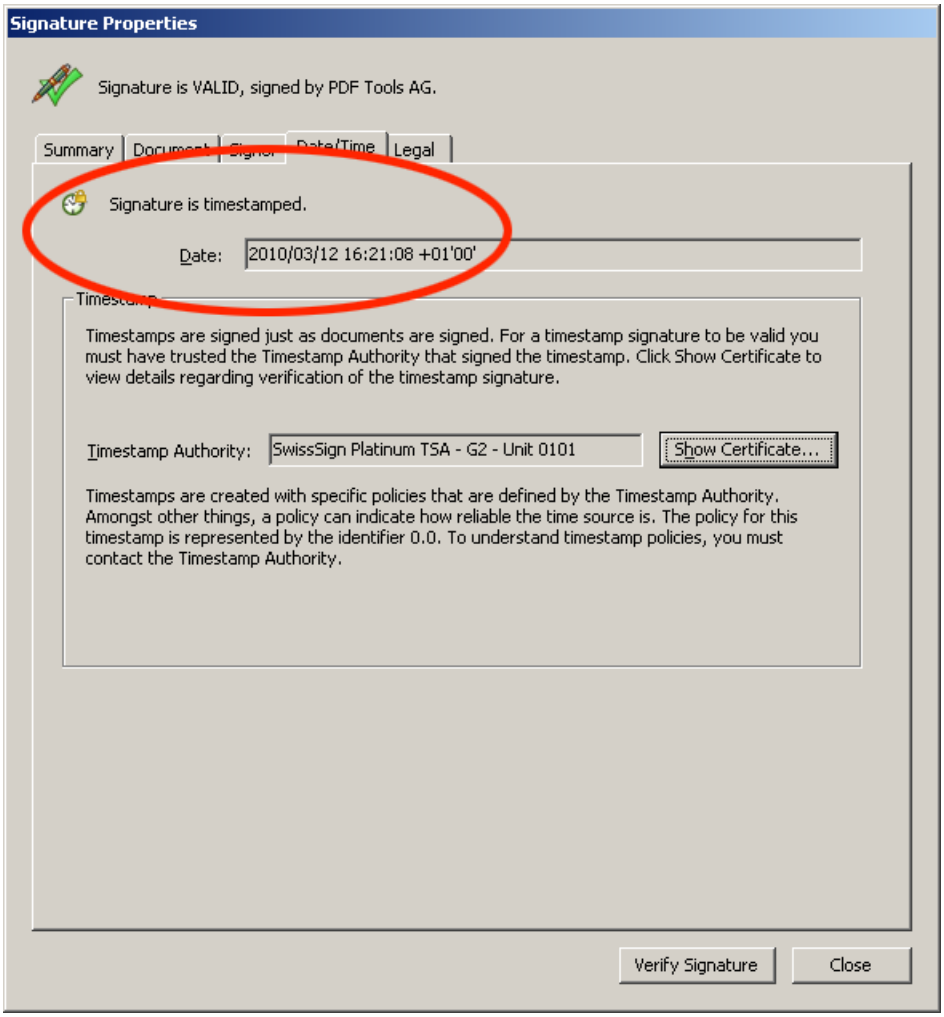
An OCSP response or CRL must be available. This is shown in the tab “Revocation”. The details should mention that “the certificate is considered *valid*”.

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



Time-stamp

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



5.7.2 Validation of a PAdES LTV Signature

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

The following must be checked:

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

Trust Chain

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

Revocation Information

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

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

Time-stamp

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

LTV Expiration Date

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

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

Other PAdES Requirements

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

See method `ValidateSignature` of the `PdfSecure` Interface.

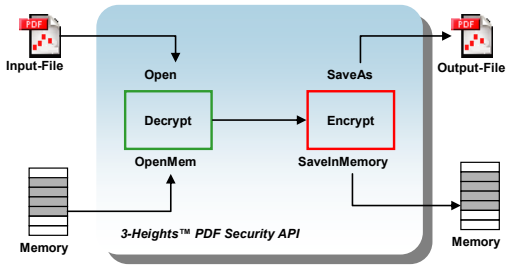
5.8 Advanced Guide

5.8.1 How to Use 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 methods `Open` and `SaveAs` are used. These two methods are described in the chapters How to read an encrypted PDF and How to encrypt a PDF.

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`, that memory block can be accessed using the method `GetPdf`.

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()
```

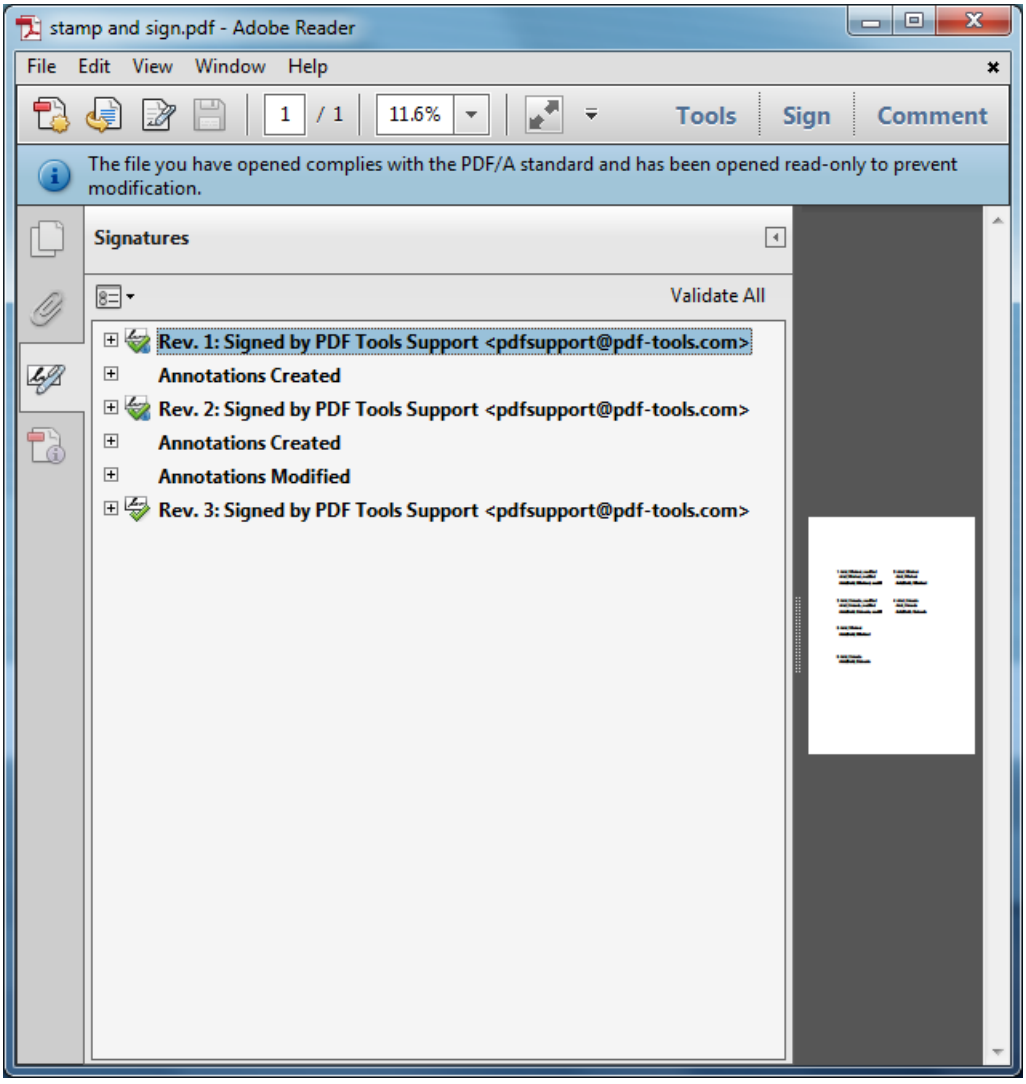
```
PDFSecure2.OpenMem(PDFSecure1.GetPdf())
PDFSecure2.SaveAs(OutputFile)
PDFSecure2.Close()
```

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.

5.9 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. In order 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.



5.9.1 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 or as memory block using the method 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.

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.
- 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.

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 a A5 and twice as large on a A3 page as specified.

`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`[12]: 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 compliance, the flag `Print` must be set and `NoView` must not be set.

Default: `Print, ReadOnly, Locked`

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 replacedm as shown in example Example 2: Modify "Simple Stamp". Note that when updating a stamp, its position and size remains. 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` and `size`.

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 overriden in `<span>`:

```
<text font="Arial" size="12" >Text with a <span
  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 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[13].

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

[13] 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›`.

`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 are `true` and `false` (**default**). See chapter Variable Text for documentation.

`text="‹text›"` (**required**)
The text that is to be written, e.g. `text="Hello World"`

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

`<span>` Define Formatting of Text

Example: `<text font="Arial" size="8"><span font="Arial,Bold">Note:</span> Text can be formatted using <span>.</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"`
- 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`. If the name is used, the respective font must be available in any of the font directories (see chapter Fonts).

`size="‹n›"` (**required**)
The font size in points, e.g. `12`. If set to `0`, the size is chosen such that text fits stamp size (not allowed if operator is within transformation operator).

`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 attribute `fontencoding` controls the subsetting:

- `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 are 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/" >PDF Tools AG</link> – Premium PDF Technology</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 the `format="‹b›"` attribute to `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: {{.

Date Values

‹value› The following values are supported:

- UTC: the current time in UTC.
- LocalTime: the current local time

‹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:

- PageCount: the number of pages in the document.

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

Example: Stamp the page count.

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

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 the 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[14], e.g. C:/pictures/image1.jpg or http://www.mydomain.com/image1.jpg.

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

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

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`

5.9.2 Examples

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.

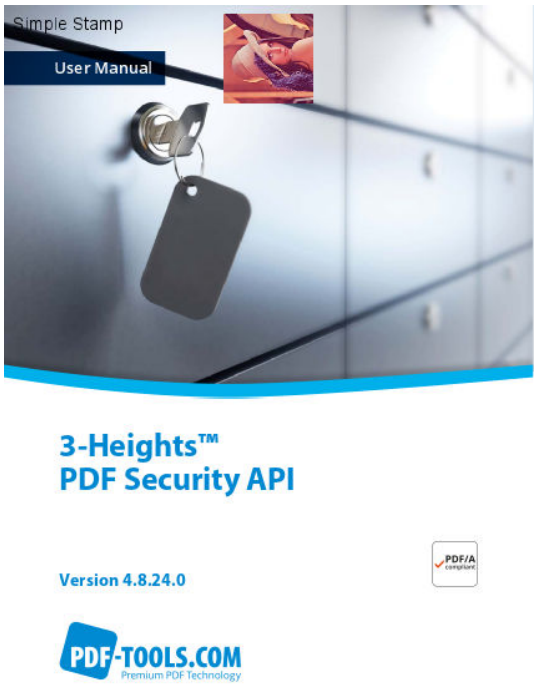
example1.xml:

```
<?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.

Example 2: Modify "Simple Stamp"

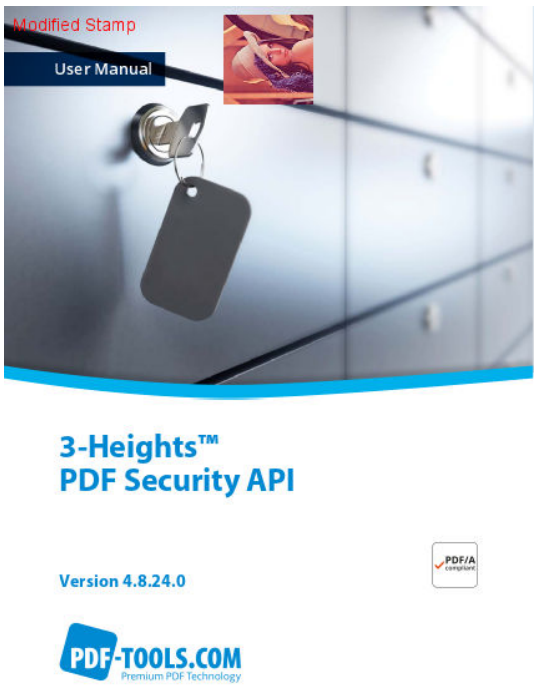
Modify `"simple stamp"` from Example 1: Simple Stamps.

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>
```



Result of example2.xml.

Example 3: Add watermark text diagonally across pages

The stamp is specified for an A4 page, which is 595 by 842 points. On each page the stamp is applied to, it is scaled (`scale="relToA4"`) and rotated (`autoorientation="true"`) to fit the page.

example3.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<pdfstamp xmlns="http://www.pdf-tools.com/pdfstamp/">
  <stamp page="all" size="595 842"
   align="center middle"
   scale="relToA4" autoorientation="true"
   type="foreground">
    <rotate angle="55" origin="298 421">
      <text mode="stroke"
       align="center middle" position="298 421"
       font="Arial,Bold" size="60"
       text="WATERMARK TEXT"/>
    </rotate>
  </stamp>
</pdfstamp>
```

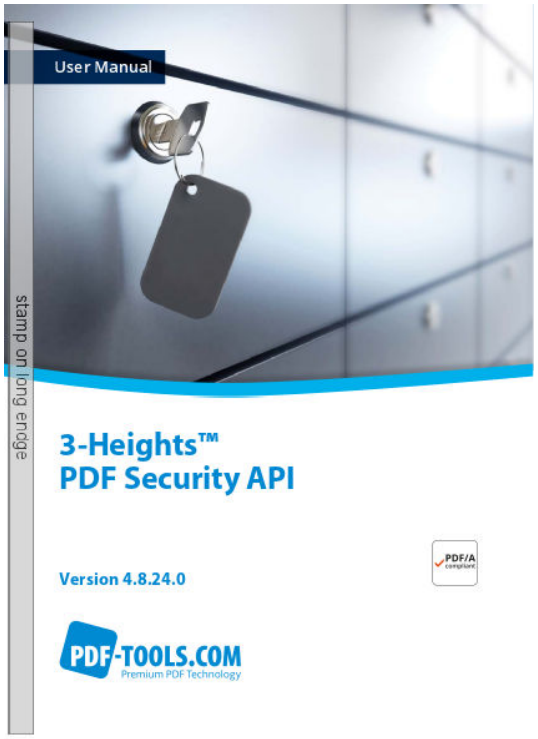
Result of example3.xml.

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 endge"/>
  </stamp>
</pdfstamp>
```



Result of example4.xml.

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.

5.10 Error Handling

Most methods of the 3-Heights™ PDF Security API can either succeed or fail depending on user input, 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. Which return values have to be interpreted as failures is documented in the chapter Reference Manual. To identify the error on a programmatic level, check the property ErrorCode. The property ErrorMessage provides a human readable error message, describing the error.

Example:

```
public Boolean 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.

6 Reference Manual

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

6.1 PdfSecure Interface

6.1.1 AddDocMDPSignature

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

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 property ForceSignature 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 will fail later on.

False Otherwise.

6.1.2 AddPreparedSignature

Method: `Boolean AddPreparedSignature(PdfSignature pSignature)`

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.

6.1.3 AddSignature

Method: `Boolean AddSignature(PdfSignature pSignature)`

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 Chapter How to Create Electronic Signatures.

Parameter:

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

Returns:

True Successfully added the signature to the document.

> **Note:** At this point it is not verified whether the certificate is valid or not. If an invalid certificate is provided the SaveAs function will fail later on.

False Otherwise.

6.1.4 AddSignatureField

Method: `Boolean AddSignatureField(PdfSignature pSignature)`

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.

6.1.5 AddStamps

Method: Boolean AddStamps(String FileName)

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 the section Stamping.

6.1.6 AddStampsMem

Method: Boolean AddStampsMem(Variant MemBlock)

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

6.1.7 AddTimeStampSignature

Method: Boolean AddTimeStampSignature(PdfSignature pSignature)

Add a document time-stamp. The following signature properties must be set: TimeStampURL. The following signature properties may be set: Provider, TimeStampCredentials.

PDF documents with document time-stamp 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 property ForceSignature is set to True.

6.1.8 AddValidationInformation

Method: Boolean AddValidationInformation(PdfSignature pSignature)

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 time-stamp tokens is added as well.

This requies a Cryptographic Provider which has been opened using BeginSession. All types of cryptographic providers support this method. However, this method will fail 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 enlarge the longevity of existing signatures. See section How to Create 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 enlarge 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.

6.1.9 BeginSession

Method: `Boolean BeginSession(String Provider)`

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

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

Parameter:

Provider [String] See property Provider.

Returns:

True Session started successfully.

False Otherwise.

6.1.10 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.

6.1.11 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 chapter Error Handling) . See also enumeration `TPDFErrorCode`. PDF-Tools error codes are listed in the header file `bseerror.h`. Please note that only few of them are relevant for the 3-Heights™ PDF Security API.

6.1.12 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. For correct usage, see chapter Error Handling.

> **Note:** The property is `Nothing` if no message is available.

6.1.13 EndSession

Method: `Boolean EndSession()`

Ends the open session to the security device.

See BeginSession.

6.1.14 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, in order to enable encryption of PDF/A conforming files. The compliance of the output file is downgraded to PDF.

6.1.15 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.

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 reivision they belong. These white space characters have no influence on the revision's visual appearance or content. However, they might 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 extractd from the updated file by reading all data up to the original file's size.

6.1.16 ForceSignature

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

Force signature allows DocMDP (PDF 1.6) and time-stamp signatures (PDF 2.0) on PDF/A-1 documents. The output file's version is upgraded and PDF/A compliance removed. Thus, the output file will contain the signature, but not be PDF/A-1 anymore.

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

6.1.17 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.

6.1.18 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 which 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.

6.1.19 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.

6.1.20 GetSignature

Method: `PdfSignature GetSignature(Long iSignature)`

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 property SignatureCount.

Returns:

An interface to the PdfSignature.

6.1.21 GetSignatureCount

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

Use the property SignatureCount instead.

6.1.22 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:** Note that the getter does not return values of the input document but merely those that have previously been set using InfoEntry.

Examples in Visual Basic 6:

Set the document title.

```
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'"
```

6.1.23 LicenseIsValid

Property (get): `Boolean LicenseIsValid`

Check if the license is valid.

6.1.24 Linearize

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

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

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 will be stored in object streams in the output PDF. For certain input documents this can lead to a significant increase of file size.

6.1.25 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 section on Caching of CRLs, OCSP, and Time-stamp Reponses for more information on the caches.

6.1.26 Open

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

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

Parameters:

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

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

Returns:

True The file could be successfully opened.

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

6.1.27 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.

6.1.28 PageCount

Property (get): `Long PageCount`

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

6.1.29 ProductVersion

Property (get): `String ProductVersion`

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

6.1.30 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`.

6.1.31 RemoveSignatureField

Method: `Boolean RemoveSignatureField(Pdfsignature pSignature)`

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

Note that 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.

6.1.32 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[15]:

- 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.

[15] 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.

- 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 time-stamp 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:
  - 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.

6.1.33 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 method GetPdf.

All parameters are identical to the SaveAs method.

See also chapter How to Use the in-Memory Functions.

Returns:

True The document could be saved in memory successfully.

False Otherwise.

6.1.34 SetLicenseKey

Method: Boolean SetLicenseKey(String LicenseKey)

Set the license key.

6.1.35 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.

6.1.36 SetSessionProperty

Method: `Boolean SetSessionPropertyString(String Name, String Value)`

Method: `Boolean SetSessionPropertyBytes(String Name, Variant Value)`

Provider-specific session configuration.

Properties have to be set before calling `BeginSession` and are deleted when calling `EndSession`.

Parameters:

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

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

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

6.1.37 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.

6.1.38 SignPreparedSignature

Method: `Boolean SignPreparedSignature(PdfSignature pSignature)`

Create a digital signature for an existing signature field, which was previously created using the method AddPreparedSignature. 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.

6.1.39 SignSignatureField

Method: `Boolean SignSignatureField(Pdfsignature pSignature)`

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.

6.1.40 Terminate

Method: `Void Terminate()`

Terminate all open sessions, and finalize and unload all PKCS#11 drivers. Some drivers require Terminate to be called. Otherwise, your application might crash and/or your HSM, USB token or smart card might not be unlocked.

When using the C/C++ API, 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.

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.

6.1.41 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.

6.1.42 ValidateSignature

Method: `Boolean ValidateSignature(PdfSignature pSignature)`

Validate an existing digital signature, which was previously retrieved using the method GetSignature. The component supports the verification of signatures including time-stamps 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 time-stamp, 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 Interface 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 property ErrorCode 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

Note that 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.

6.2 PdfSignature Interface

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

6.2.1 ContactInfo

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

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

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

6.2.2 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.

6.2.3 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.

6.2.4 DocumentHasBeenModified

Property (get): Boolean DocumentHasBeenModified

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

6.2.5 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.

6.2.6 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. Use this property for signatures of type adbe.pkcs7.detached (see SubFilter). For other types of signatures, 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 chapter Caching of CRLs, OCSP, and Time-stamp 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 after SaveAs is SIG_CREA_E_OCSP.

6.2.7 FillColor

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

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

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

6.2.8 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.

6.2.9 Filter

Property (get): String Filter

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

6.2.10 FontName1

Property (get, set): String FontName1
Default: "Arial"

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

6.2.11 FontName2

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

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

6.2.12 Font1Mem

Property (set): Variant Font1Mem

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

6.2.13 Font2Mem

Property (set): Variant Font2Mem

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

6.2.14 FontSize1

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

Define the font size of the Text1.

6.2.15 FontSize2

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

Define the font size of the Text2.

6.2.16 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.

6.2.17 ImageFileName

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

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

If you want the appearance to contain the image only and no text, set the property `Text2` to a space `" "`.

6.2.18 Issuer

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

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

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

6.2.19 LineWidth

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

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

6.2.20 Location

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

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

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

6.2.21 Name

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

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

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

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

When using a Windows OS, the certificate must be available in the Windows certificate store. See also chapter Digital Signatures.

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

6.2.22 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.

6.2.23 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"
```

6.2.24 ProxyURL

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

This property has been deprecated. For more information, see the chapter How to Use a Proxy.

6.2.25 ProxyCredentials

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

This property has been deprecated. For more information, see the chapter How to Use a Proxy.

6.2.26 Reason

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

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

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

6.2.27 Rect

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

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

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

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

6.2.28 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`.

6.2.29 SerialNumber

Property (get, set): `String SerialNumber`

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

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

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

6.2.30 SignerFingerprint

Property (get, set): `Variant SignerFingerprint`

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

6.2.31 SignerFingerprintStr

Property (get, set): `String SignerFingerprintStr`

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

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

6.2.32 Store

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

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

6.2.33 StoreLocation

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

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

`0` Local Machine

`1` Current User (default)

For more information, see the detailed description of the Windows Cryptographic Provider.

6.2.34 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.

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

6.2.35 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 (ETSI TS 103 172) signature used for document signatures (`AddSignature`) and DocMDP signatures (`AddDocMDPSignature`).

`ETSI.CAdES.detached` (PDF 2.0) PAdES signature as specified by new European norm ETSI EN 319 142. This type is used for document signatures (`AddSignature`) and DocMDP signatures (`AddDocMDPSignature`). See chapter How to Create a PAdES Signature for more information.

6.2.36 Text1

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

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

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

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

‹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"
```

6.2.37 Text2

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

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

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

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

See also property `Text1`. If you want the appearance to not contain any text, set this property to a space `" "`.

6.2.38 TimeStampCredentials

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

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

6.2.39 TimeStampFingerprint

Property (get): `Variant TimeStampFingerprint`

The sha1 fingerprint of the time-stamp server certificate. After validating a signature that contains a time-stamp, this property contains the fingerprint of the time-stamp server's certificate.

6.2.40 TimeStampURL

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

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

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

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

6.2.41 UserData

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

This property has only a meaning if a Custom Signature Handler is used.

6.3 Enumerations

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

6.3.1 TPDFErrorCode

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

The single letter gives an indication of the severity of the error. These are: Success, Error, Warning and Information. In general, an error is returned if an operation could not be completed, e.g. no valid output file was created. A warning is returned if the operation was completed, but problems occurred in the process.

A list of all error codes is available in the C API's header file `bseerror.h`, the javadoc documentation of `com.pdftools.NativeLibrary.ERRORCODE` and the .NET documentation of `Pdftools.Pdf.PDFErrorCode`. Note that only a few are relevant for the 3-Heights™ PDF 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.
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 compliance. 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.
SIG_CREA_E_SESSION	Cannot create a session (or CSP).
SIG_CREA_E_STORE	Cannot open certificate store.
SIG_CREA_E_CERT	Certificate not found in store.
SIG_CREA_E_INVCERT	The signing certificate is invalid.
SIG_CREA_E_OCSP	Couldn't get response from OCSP server.
SIG_CREA_E_CRL	Couldn't get response from CRL server.
SIG_CREA_E_TSP	Couldn't get response from time-stamp server.

SIG_CREA_E_PRIVKEY	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 time-stamp is invalid.
SIG_VAL_W_TSPCERT	The time-stamp 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.[16]
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 time-stamp.
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.

6.3.2 TPDFPermission

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 shall not be combined with another enumeration. When using this enumeration set both passwords to an empty string or Nothing.
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
ePermAssemble	Document assembly

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

TPDFPermission Table

ePermDigitalPrint	High resolution printing
ePermAll	Grant all permissions

Changing permissions or granting multiple permissions is done using a bitwise or operator. 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
```

7 Version History

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

7.1 Changes in Version 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 time-stamps.
  - **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.
- Writing PDF objects into object streams is now supported. 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.
- Increased 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.

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`.

7.2 Changes in Version 4.9

- 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.
- Support OpenType font collections in installed font collection.
- Improved metadata generation for standard PDF properties.
- [C] **Changed** return value `pfGetLength` of `TPDFStreamDescriptor` to `pos_t`[17].

Interface `PdfSecure`

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

7.3 Changes in Version 4.8

- 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.
- The creation of annotation appearances has been optimized 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`.

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

8 Licensing, Copyright, and Contact

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

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

Licensing and Copyright

Contact

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