

PDFSignDll for .NET Framework SDK User Manual

Introduction

The main function of PDFSignDll SDK is to sign and encrypt PDF documents using X.509 certificates. Using this library you can quickly sign and encrypt PDF files using a fully configurable appearance. The library is compiled using Microsoft .NET Framework 2.0.

The positioning of the signature appearance is configurable, plus on which pages of the document it should appear (first page, last page or all pages).

Licensing

Internal license is used by companies (or freelancers) which need PDFSignDll for internal company use, such as custom digital signature software.

Web license is required when you use PDFSignDll in web applications. This license can be used on any number of systems in one company.

Company license is offered to software vendors (freelancers or companies), who develop their software for third-parties (companies or individuals). If you develop a software that will be distributed to other companies or individual users, then you need a Company license.

All licenses can be used by any number of developers from the company.

Links

PDFSignDll main page: <http://www.signfiles.com/pdf-sign-library/>

PDFSignDll Demo library (.DLL only): <http://www.signfiles.com/sdk/PDFSignDll.zip>

Code example: <http://www.signfiles.com/pdf-sign-library-code-example/>

PDFSignDll SDK Quick Manual: <http://www.signfiles.com/pdfsingndll-quick-manual/>

Warning and Disclaimer

Every effort has been made to make this manual as complete and accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this manual.

Trademarks

.NET, Visual Studio .NET are trademarks of Microsoft Inc.

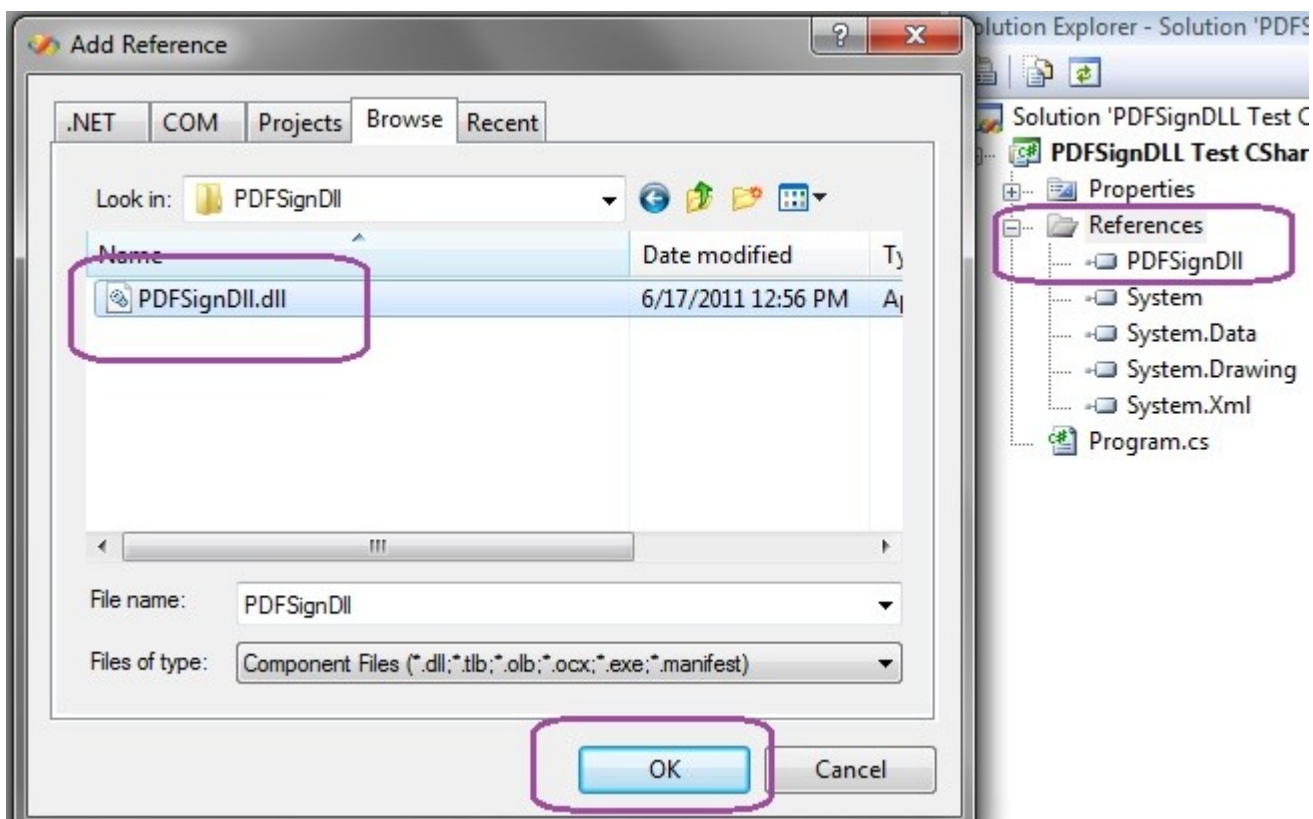
Adobe, Adobe Reader are trademarks of Adobe Systems Inc.

All other trademarks are the property of their respective owners.

Introduction	1
How to use the PDFSignDll library in Visual Studio.....	3
Digital Certificates.....	4
Digital certificates used for digital signatures.....	4
Digitally sign a PDF file using a digital certificate stored on PFX file.....	5
Digitally sign a PDF file using a digital certificate stored on Microsoft Store.....	6
Validating Digital Signatures in Adobe.....	7
Digitally sign a PDF with PDFSignDLL.....	8
Loading the PDF.....	8
Digitally sign an encrypted PDF file.....	8
Obtain document information (number of pages, page size).....	8
Set digital signature properties (reason, location).....	9
Set the digital signature rectangle properties.....	10
Set a custom digital signature text.....	10
Set the text direction on the signature rectangle.....	10
Set the digital signature font.....	11
Set the digital signature graphic.....	11
Set a visible or hidden signature.....	11
Time Stamping.....	12
Time stamp the PDF digital signature.....	12
Authentication with Username and Password.....	12
Authentication with a digital certificate.....	13
Nonce and time stamping policies OID.....	13
Proxy settings.....	13
Validating the time stamping response on Adobe.....	14
Certify a PDF digital signature.....	15
Digital signatures and PDF/A standard.....	16
Other features of the library.....	17
Digitally sign all pages from a PDF document.....	17
Adding multiple digital signatures on the PDF document.....	17
Set an approximate block size for the digital signature.....	17
Old style Adobe digital signature appearance.....	18
Include the signing certificate chain on the signature.....	19
Hash algorithms.....	19
Verifying the signing certificate.....	19
Digital signature and encryption.....	20
Password security.....	20
Digital Certificate security.....	22
Code snippets.....	24
Digitally sign all pages from a PDF file with a certificate stored on PFX file.....	24
Set a custom signature rectangle and sign using a certificate from Microsoft Store.....	24
Digitally sign a PDF located on the web only if it is not already signed.....	24
Set a custom text and font for the digital signature rectangle.....	25
Add an image on the signature rectangle and save the file as PDF/A.....	25
Set an invisible signature and certify a PDF file.....	26
Time stamp a PDF file.....	26
Time stamp a PDF file using authentication.....	26
Time stamp a PDF file using a proxy connection and set the TSA Policy OID.....	27
Digitally sign and time stamp a folder with PDF files.....	27
Verify a digital signature.....	28
Merge multiple PDF files into a single PDF file.....	29
Insert texts and images in a PDF file.....	29
Exceptions.....	31
License Agreement (EULA).....	33

How to use the PDFSignDII library in Visual Studio

- Download the library (or the demo project) from <http://www.signfiles.com/pdf-sign-library/>
- Unzip the file and copy the *PDFSignDII.dll* and *PDFSignDII.xml* on your project location.
- In your project, go to *References*, select *Add Reference...*, select the *PDFSignDII.dll* as below.
- Note that all methods, properties, enums and classes are described on the file *PDFSignDLL help file.chm*



Adding as reference PDFSignDII library

Digital Certificates

Digital certificates used for digital signatures

To digitally sign a PDF file a digital certificate is needed. The digital certificates are stored in two places:

- in Microsoft Store
- in PFX or P12 files

The certificates stored on **Microsoft Store** are available by opening *Internet Explorer – Tools* menu – *Internet Options* – *Content* tab – *Certificates* button (see below).

For digital signatures the certificates stored on *Personal* tab are used. These certificates have a public and a private key.

The digital signature is created by using the private key of the certificate. The private key can be stored on the file system (imported PFX files), on an cryptographic smart card (like Aladdin eToken or SafeNet iKey) or on a HSM (Hardware Security Module).

For encryption, only the public key of the certificate is necessary (certificates stored on *Personal* or *Other People* tabs).



Signing certificates available on Microsoft Store

Another way to store a digital certificate is a **PFX (or P12) file**. This file contains the public and the private key of the certificate. This file is protected by a password in order to keep safe the key pair.

Note that a PFX file can be imported on Microsoft Store (just open the PFX file and follow the wizard).

To obtain a digital certificate (in PFX format) follow this link:

<https://ca.signfiles.com/userEnroll.aspx>

Digitally sign a PDF file using a digital certificate stored on PFX file

The code below demonstrates how to digitally sign a PDF file using a PFX certificate.

```
//Initializes a new instance of PDFSign class
PDFSign PDFSign = new PDFSign("Enter your application license here");

//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

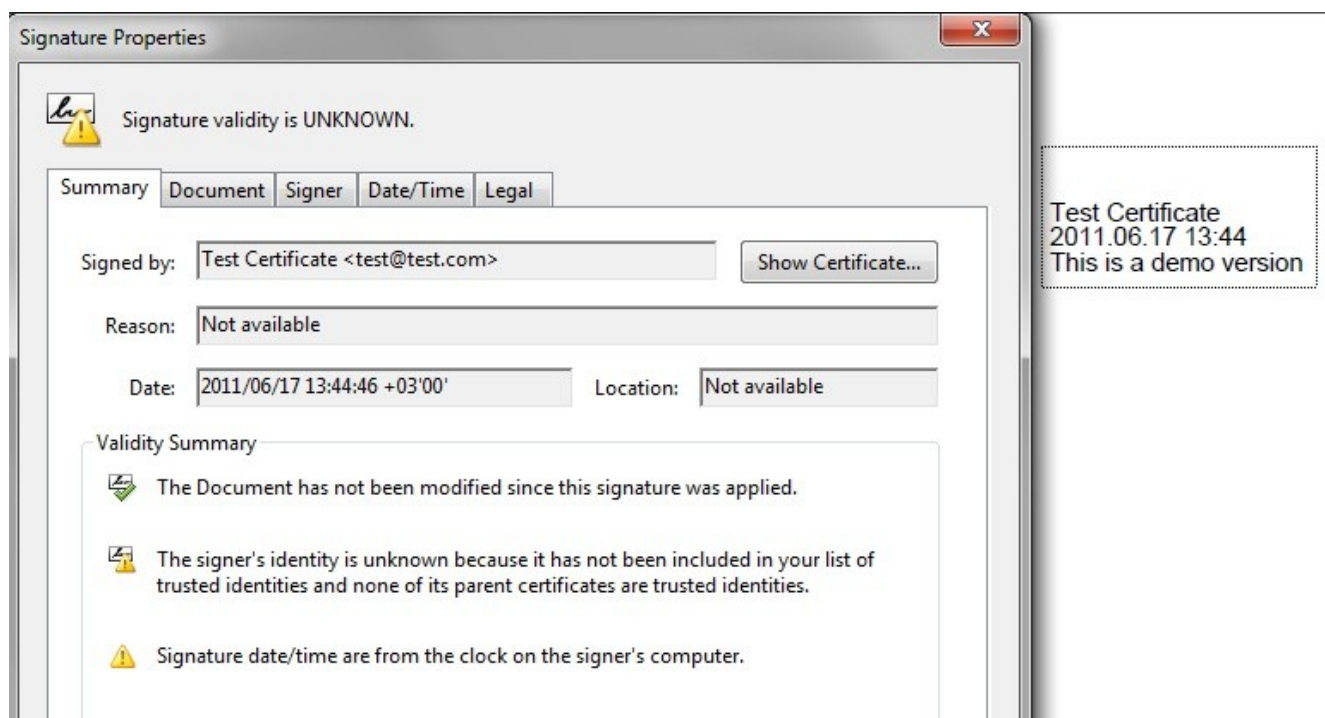
//Load the certificate from .PFX
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "PFX certificate
password");

//Set the signature rectangle attributes
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopRight;

//digitally sign and save the PDF file
File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

When the *dest.pdf* is opened in Adobe Reader, a signature rectangle appear on the top right corner.

When the signature rectangle is clicked, the digital signature information appears.



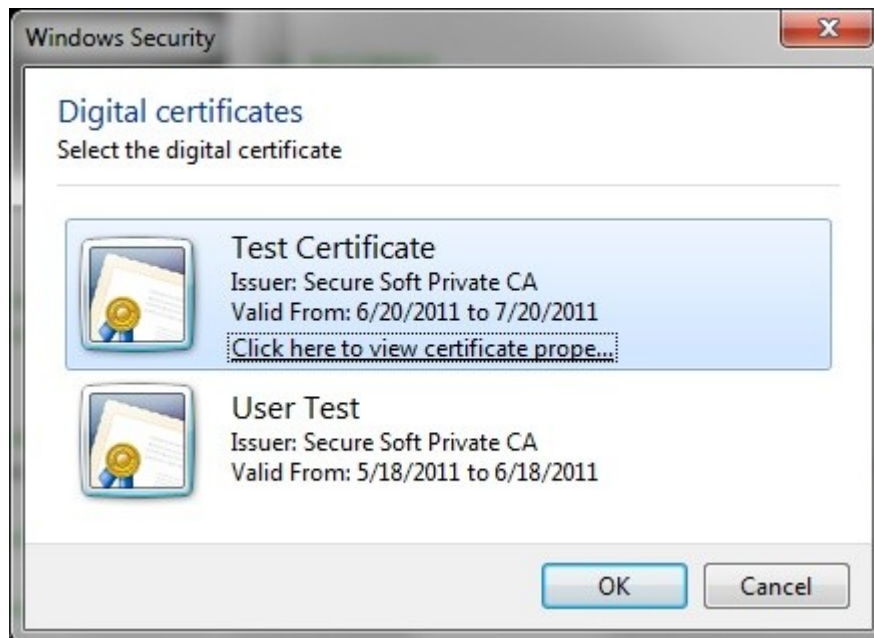
Digital signature properties on Adobe Reader

Digitally sign a PDF file using a digital certificate stored on Microsoft Store

To digitally sign a PDF using a certificate stored on Microsoft store use this line of code:

```
PDFSign.DigitalSignatureCertificate = PDFSign.LoadCertificate(false, "", "Digital certificates", "Select the digital certificate", DigitalCertificateScope.ForDigitalSignature);
```

When the project is launched, the user must select the digital certificate from all certificates available in *Personal* tab.



Digital certificates selection window

Probably you want to digitally sign a PDF file without user intervention using a certificate available on Microsoft Store. For that you must identify that certificate by a criteria.

If your desired certificate has in the *Subject* filed the value $E = email@email.com$, you can use the following code to automatically use this certificate for the signing operation.

```
PDFSign.DigitalSignatureCertificate = PDFSign.LoadCertificate(false, DigitalCertificateSearchCriteria.Email, "test@test.com", DigitalCertificateScope.ForDigitalSignature);
```

Note that there are a lot of criteria to automatically select your certificate (Common Name, Serial number, Thumbprint, etc.).

Validating Digital Signatures in Adobe

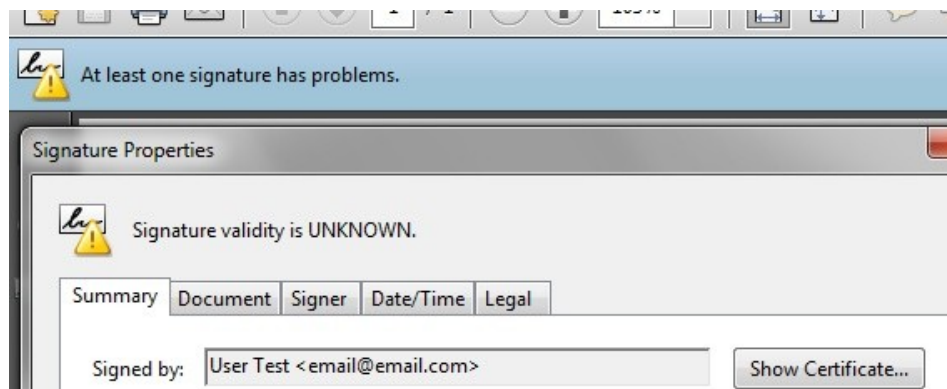
Every digital certificate is issued by a Root CA (Certification Authority). Some of the Root CA's are included by default in Windows Certificate Store (Trusted Root Certification Authorities) and only a few are included in Adobe Certificate Store. Microsoft and Adobe use different Certificate Stores different certificate validation procedures.

If the signing certificate (or the Root CA that issued the signing certificate) is not included in Adobe Store, the digital signature is considered "not trusted" when a user open a document with Adobe Reader (see example).

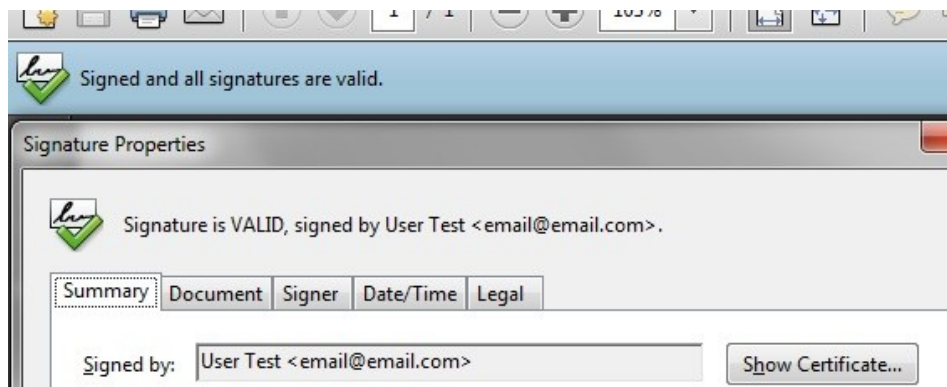
This behavior has nothing to do with the signing engine but with the Adobe certification validation procedure.

To trust a signature the user must add the signing certificate on the Adobe Certificate Store because only a few Root CA's are considered trusted by default by Adobe certificate validation engine (See this article: http://www.adobe.com/security/partners_cds.html)

To validate the signing certificate in Adobe use the methods described on this document: <http://www.signfiles.com/manuals/ValidatingDigitalSignaturesInAdobe.pdf>



Validity Unknown signature



Valid signature

Digitally sign a PDF with PDFSignDLL

Loading the PDF

The PDF can be loaded from a file, a byte array or from an URL like below:

```
//Load the PDF from byte[] array
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

//Load the PDF from a file
PDFSign.LoadPDFDocument("c:\\source.pdf");

//Load the PDF from an URL
PDFSign.LoadPDFDocument(new Uri("http://www.signfiles.com/test.pdf"));
```

Digitally sign an encrypted PDF file

To digitally sign an encrypted PDF file you must first provide the protection password like below:

```
//Initializes a new instance of PDFSign class
PDFSign PDFSign = new PDFSign("Enter your application license here");

//set the document password first
PDFSign.DocumentProperties.Password = "document password";

//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));
```

Obtain document information (number of pages, page size)

In some cases you will need some information about the opened document (is document already signed, number of pages, document page size).

DocumentPageSize property is useful when you want to place a custom digital signature rectangle on the PDF document.

DocumentProperties.NumberOfPages is useful when you want to place a signature on the last page of the document.

```
//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

//get the page size of the last page of the document
PDFSign.DocumentPageSize(PDFSign.DocumentProperties.NumberOfPages);

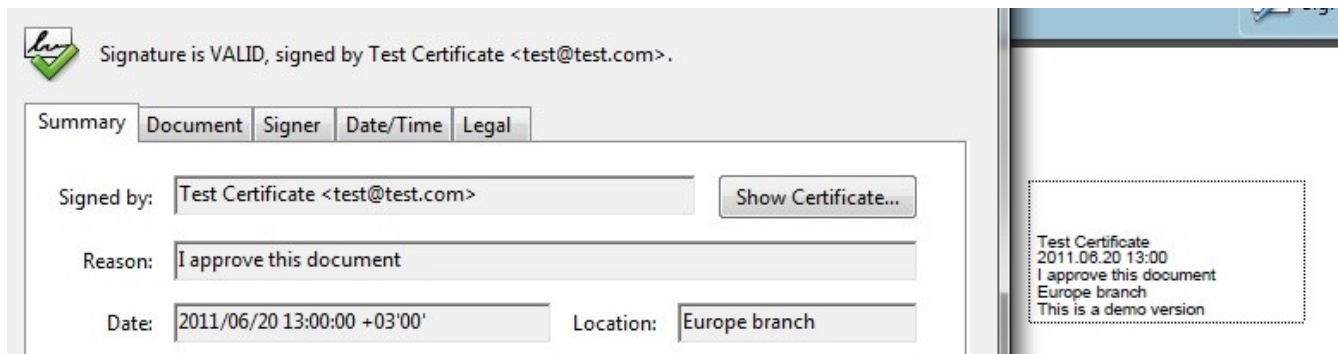
//get the number of digital signatures already attached to this document
int signatures = PDFSign.DocumentProperties.NumberOfDigitalSignatures;
```

Set digital signature properties (reason, location)

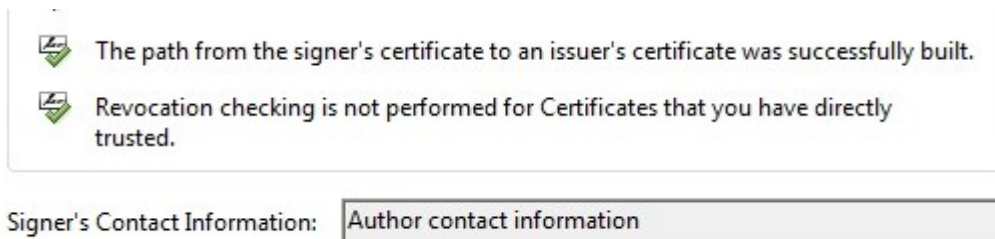
Adobe digital signatures can be fully customized by PDFSignDLL SDK. In order to set the Reason, Location, Date, "Signed by" or "Signer's Contact Information" property use the code below.

Observation: Some digital signature properties (like "Signed by" in Adobe) will not appear with your custom value because of Adobe policy. If Time stamping is used, the signing date (*SignatureDate* property) is taken from the time stamping response.

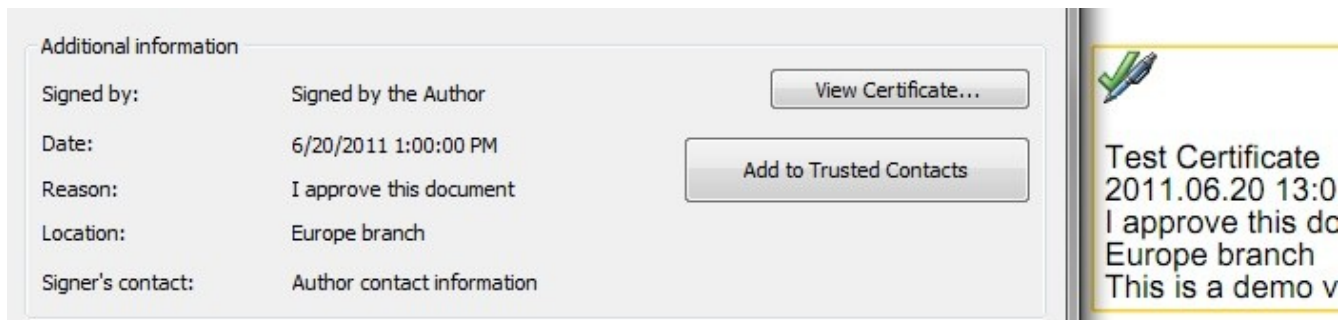
```
PDFSign.SigningReason = "I approve this document";  
PDFSign.SigningLocation = "Europe branch";  
PDFSign.SignerContactInformation = "Author contact information";  
PDFSign.SignedBy = "Signed by the Author";  
PDFSign.SignatureDate = new DateTime(2011, 6, 20, 13, 00, 00);
```



Signed by, Reason, Location and Date properties in Adobe



Signer's Contact Information in Adobe



Signed by, Reason, Location, Date and Signer's contact properties in other PDF reader

Set the digital signature rectangle properties

The digital signature rectangle can appear on the PDF document on a standard location (like Top Right) or in a custom place based on the PDF page size.

Example: put the digital signature rectangle on the last page of the document on top middle position:

```
PDFSign.SignaturePage = PDFSign.DocumentProperties.NumberOfPages;  
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopMiddle;
```

Observation: In Adobe, the corner (0,0) is on the bottom left of the page.

Example: put the digital signature on a custom position (top right corner) on the first page of the document:

```
PDFSign.SignaturePage = 1;  
//get the pdf page size  
System.Drawing.Point page = PDFSign.DocumentPageSize(1);  
  
//set the rectangle width and height  
int width = 80;  
int height = 40;  
PDFSign.SignaturePosition = new System.Drawing.Rectangle(page.X - width, page.Y -  
height, width, height);
```

Set a custom digital signature text

The default digital signature text contains information extracted from the signing certificate, signing date, signing reason and signing location.

If you are using a demo version of the library, the text *"This is a demo version"* is also appended. This text will be removed on the registered version of the library.

The signature text can be set using *SignatureText* property like below:

```
PDFSign.SignatureText =  
"Signed by:" +  
PDFSign.DigitalSignatureCertificate.GetNameInfo(X509NameType.SimpleName, false) +  
"\n Date:" + DateTime.Now.ToString("yyyy.MM.dd HH:mm") + "\n" +  
"Reason:" + PDFSign.SigningReason;
```

Set the text direction on the signature rectangle

The default text direction is left to right. To change the text direction to right to left use the following code (e.g. for Hebrew language):

```
PDFSign.TextDirection = TextDirection.RightToLeft;
```

Set the digital signature font

The default font file for the digital signature rectangle is Helvetica. It is possible that this font to not include all necessary UNICODE characters like ä, à, â. On this case you will need to use an external font.

The font size is calculated based on the signature rectangle size in order to fit on the signature rectangle (it not have a fixed size). To set the font size you can use *FontSize* property like below:

```
PDFSign.FontFile = "c:\\windows\\fonts\\arial.ttf";
PDFSign.FontSize = 10;
```

Set the digital signature graphic

The digital signature rectangle can contains text, graphic or text with graphic. To add an image on the digital signature rectangle use the following code:

```
PDFSign.SignatureText = "Signed by the Author";
PDFSign.SignatureImage = System.IO.File.ReadAllBytes("c:\\graphic.jpg");

//text on the right and image on the left
PDFSign.SignatureImagePosition = SignatureImageType.ImageAndText;
//image as bakground and text on above
PDFSign.SignatureImagePosition = SignatureImageType.ImageAsBackground;
//only image
PDFSign.SignatureImagePosition = SignatureImageType.ImageWithNoText;
```

These types of signatures are shown below:



1. Image and text, 2. Image as background, 3. Image with no text

Set a visible or hidden signature

Sometimes the digital signature rectangle is not necessary to appear on the PDF document. The default value of *VisibleSignature* property is true.

To set an invisible digital signature use the code below:

```
//invisible signature
PDFSign.VisibleSignature = false;

//digitally sign and save the PDF file
File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

Time Stamping

Time stamp the PDF digital signature

Timestamping is an important mechanism for the long-term preservation of digital signatures, time sealing of data objects to prove when they were received, protecting copyright and intellectual property and for the provision of notarization services.

To add time stamping information to the PDF digital signature you will need access to a [RFC 3161](#) time stamping server.

A fully functional version of our TSA Authority is available for testing purposes at this link: <http://ca.signfiles.com/TSAServer.aspx> (no credentials are needed).

Use the code below to digitally sign and timestamp your PDF file:

```
//Initializes a new instance of PDFSign class
PDFSign PDFSign = new PDFSign("Enter your application license here");

//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

//Load the certificate from .PFX
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");

//Set the signature rectangle attributes
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopRight;

//Time stamp the PDF digital signature
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");

//digitally sign and save the PDF file
File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

Authentication with Username and Password

If your TSA server requires username and password, use the following code:

```
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");
PDFSign.TimeStamping.UserName = "username";
PDFSign.TimeStamping.Password = "password";
```

Authentication with a digital certificate

In some cases, the access to your **TSA server must be done using a digital certificate (authentication certificate)**. On this case use the following code:

```
//Time stamp the PDF digital signature
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");

PDFSign.TimeStamping.AuthenticationCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\time_stamping_certificate.pfx"),
"123456");
```

Nonce and time stamping policies OID

The **nonce**, if included, allows the client to verify the timeliness of the response when no local clock is available. The nonce is a large random number with a high probability that the client generates it only once (e.g., a 64 bit integer).

To include (or exclude) a Nonce on the time stamping request use the following code. The default value of the *UseNonce* property is true.:

```
PDFSign.TimeStamping.UseNonce = true;
```

Some TSA servers require to set a **Policy OID** on the TSA requests. To set a TSA policy OID on the time stamping requests use the code below. By default, no TSA OID is included on the TSA request.

```
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");

PDFSign.TimeStamping.PolicyOID = new
System.Security.Cryptography.Oid("1.3.7.2.9.1.829.3");
```

Proxy settings

If you are behind a Proxy server, a TSA request can be done using the following code:

```
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");

//Code example:
//http://msdn.microsoft.com/en-us/library/system.net.httpwebrequest.proxy.aspx

//set the proxy settings
System.Net.WebProxy tsaProxy = new System.Net.WebProxy();
tsaProxy.Address = new Uri("http://myproxy.example.com:1024");
tsaProxy.Credentials = new System.Net.NetworkCredential("username", "password");

PDFSign.ProxySettings = tsaProxy;
```

Validating the time stamping response on Adobe

As digital signatures certificates, the time stamping responses are signed by a certificate issued by a Certification Authority.

If the time stamping certificate (or the Root CA that issued the time stamping certificate) is not included in Adobe Store, the time stamping response could not be verified when a user open a document with Adobe Reader (see example).

This behavior has nothing to do with the signing engine but with the Adobe certification validation procedure.

To validate the signing certificate in Adobe use the methods described on this document: <http://www.signfiles.com/manuals/ValidatingDigitalSignaturesInAdobe.pdf>.



Not verified timestamp



Trusted time stamping response

Certify a PDF digital signature

When you certify a PDF, you indicate that you approve of its contents. You also specify the types of changes that are permitted for the document to remain certified.

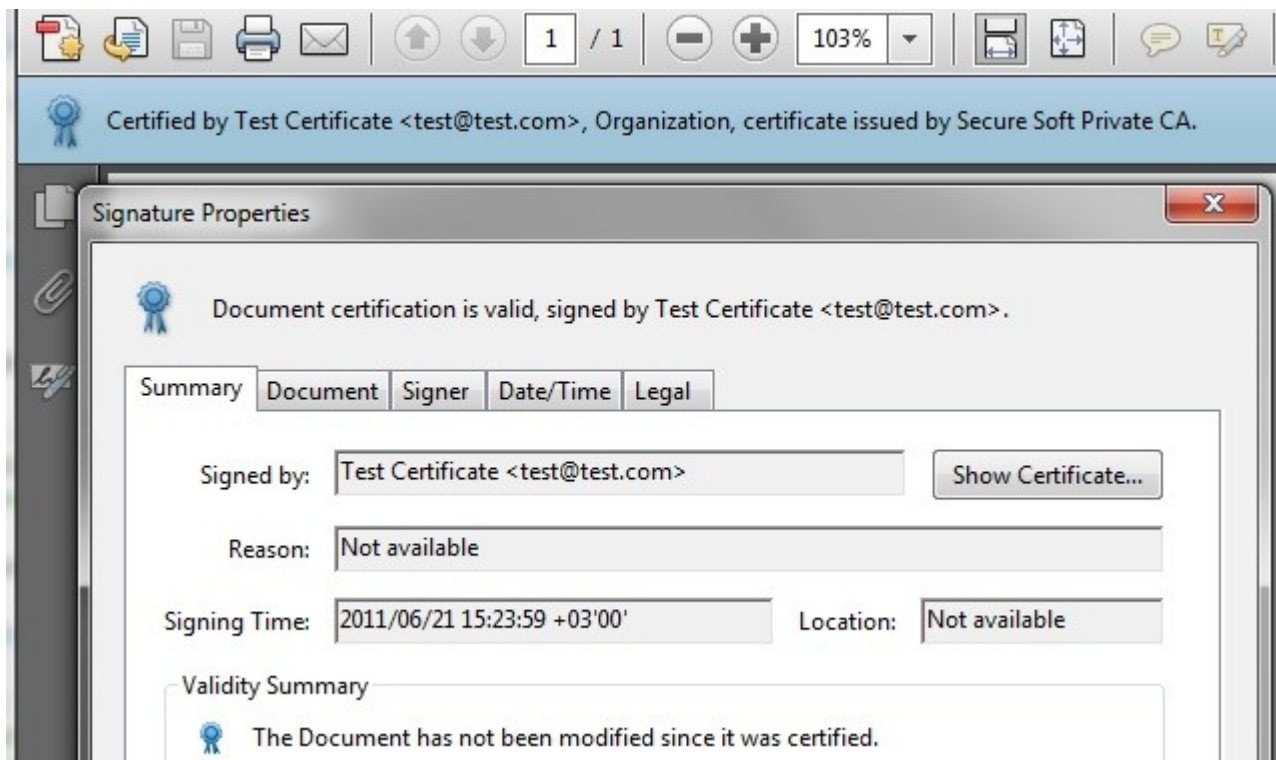
You can apply a certifying signature only if the PDF doesn't already contain any other signatures. Certifying signatures can be visible or invisible. A blue ribbon icon in the Signatures panel indicates a valid certifying signature (see example).

More information about the certification process you can find [here](#).

To certify a digital signature use the following code:

```
//adding annotations and form filling are allowed
PDFSign.CertifySignature = CertifyMethod.AnnotationsAndFormFilling;
//form filling is allowed
PDFSign.CertifySignature = CertifyMethod.FormFilling;
//no changes allowed
PDFSign.CertifySignature = CertifyMethod.NoChangesAllowed;

//digitally sign and save the PDF file
File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```



Certified signature

Digital signatures and PDF/A standard

PDF/A is a file format for the long-term archiving of electronic documents. It is based on the PDF Reference Version 1.4 from Adobe Systems Inc. (implemented in Adobe Acrobat 5 and latest versions) and is defined by ISO 19005-1:2005.

PDFSignDLL library can save PDF file in PDF/A-1b - Level B compliance in Part 1 standard.

Observation: In order to save a PDF/A-1b file all fonts used on the PDF document must be embedded (including the font used on the digital signature rectangle).

To digitally sign a file in PDF/A-1b standard use the following code:

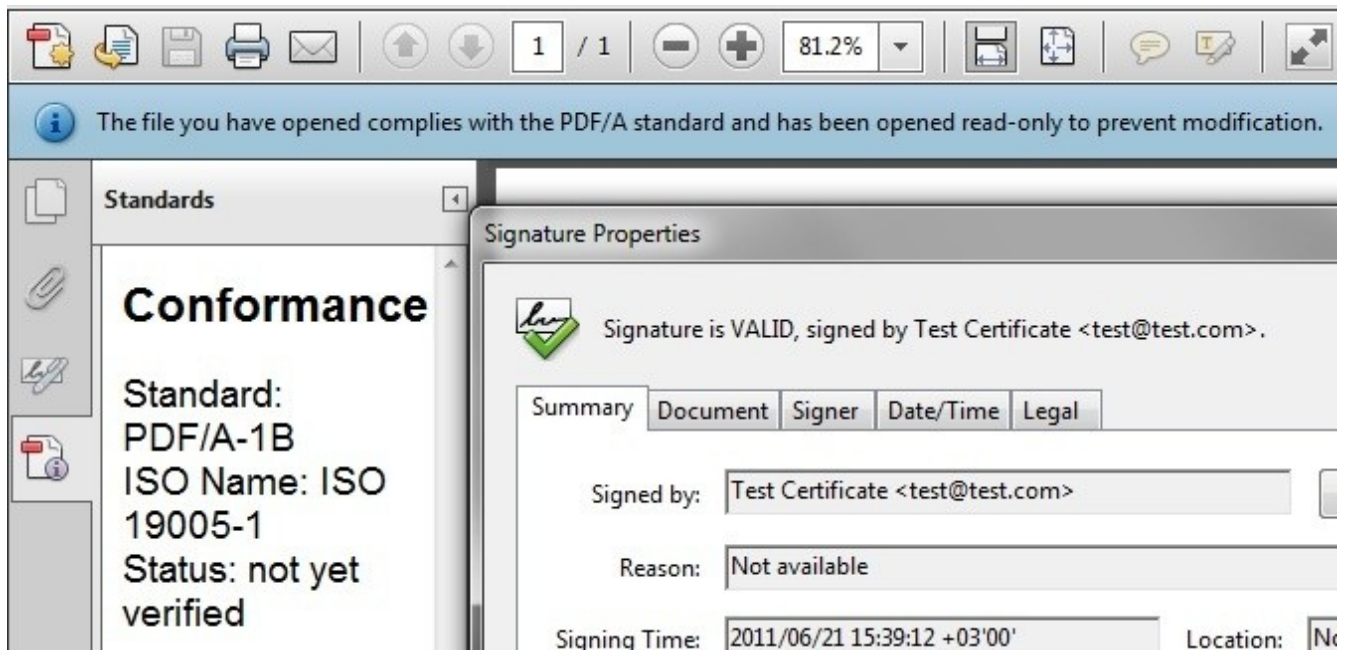
```
//Initializes a new instance of PDFSign class
PDFSign PDFSign = new PDFSign("Enter your application license here");

//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

//Load the certificate from .PFX
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopRight;

PDFSign.SaveAsPDFA = true;
PDFSign.FontFile = "c:\\windows\\fonts\\arial.ttf";

//digitally sign and save the PDF file
File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```



PDF/A-1b document with digital signature

Other features of the library

Digitally sign all pages from a PDF document

To add the digital signature rectangle to all pages from the PDF document use the following code (the default values is false):

```
PDFSign.SignaturePage = 1;  
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopRight;  
  
PDFSign.SignatureToAllPages = true;
```

Adding multiple digital signatures on the PDF document

Digital signature is appended to the document in order to add multiple signatures to the document. In order to add only one digital signature set the *AppendSignature* property to false (the default value is true). When you choose to encrypt and digitally sign a PDF file *AppendSignature* property will be automatically set to false.

This is an invisible property and will not appear on autocomplete.

```
PDFSign.AppendSignature = false;
```

Set an approximate block size for the digital signature

The default block size for the digital signature information is 16384 bytes. This space should be enough for the digital signature information and the time stamping response.

In some cases, the size of the document is an critical factor so the size of the signed file can be reduced by setting a lower value of the signature block size.

Observation: This value is approximative and cannot be set on the signed document to an exact value so the final size of the signed file is not equal with the original file size + *SignatureByteBlockSize*.

The digital signature block contains:

- public key of the signing certificate
- information like signing reason, signing location
- document signed digest in PKCS#7 format
- time stamping response

To set a custom space for the signature block size (**this is an invisible property and will not appear on autocomplete**) use the following code:

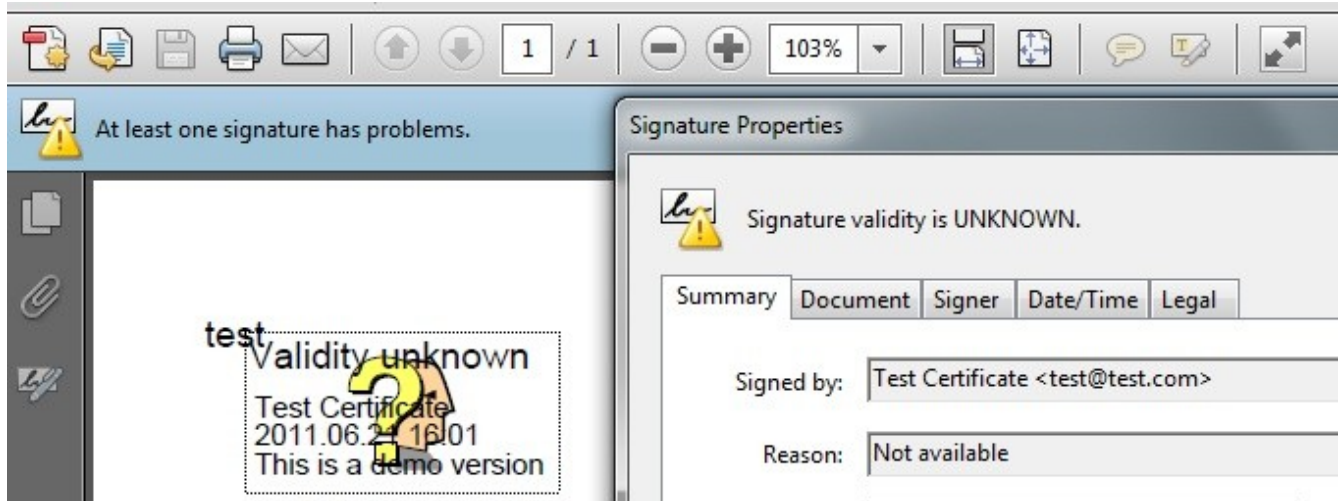
```
PDFSign.SignatureByteBlockSize = 8192;
```

Old style Adobe digital signature appearance

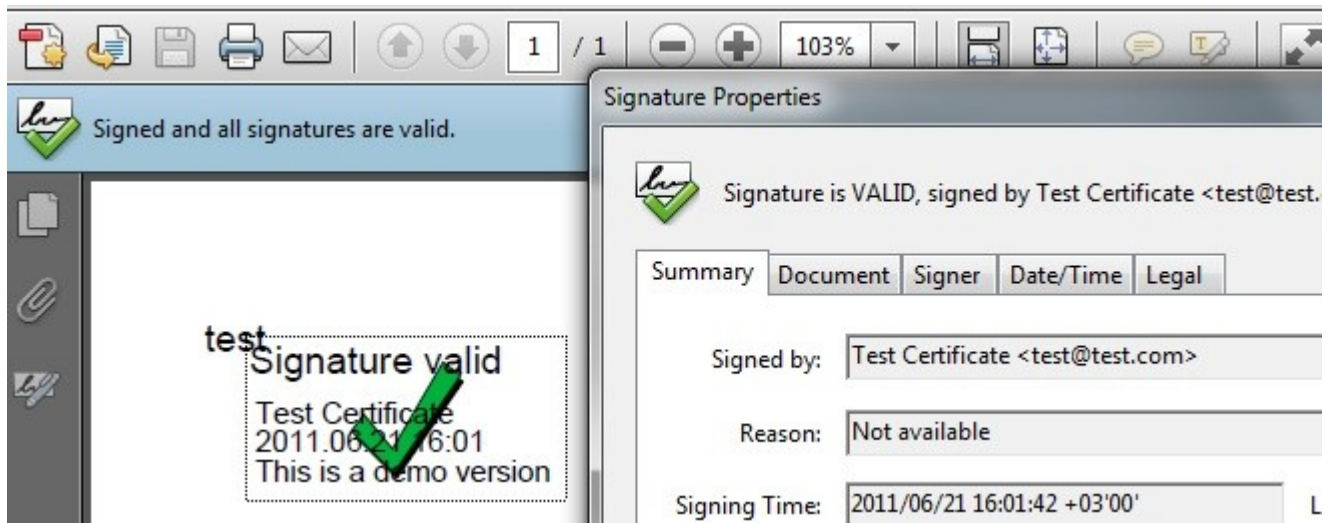
To use an old style appearance of the digital signature rectangle (see example) set the *OldStyleAdobeSignature* property to true. The default value is false.

This is an invisible property and will not appear on autocomplete.

```
PDFSign.OldStyleAdobeSignature = true;
```



Validity unknown signature



Signature valid

Include the signing certificate chain on the signature

If the chain of the signing certificate is available it will be included on the signature. The chain of a PFX/P12 certificate is not included on the signature.

To include only the signing certificate and not the entire chain, use the following code:

```
PDFSign.IncludeWholeChain = false;
```

Hash algorithms

The default (and recommended) hash algorithm used by the library is **SHA1** but in some cases, SHA256/384/512 must be used (note that not all PDF readers can validate a such of signature). To set the hash algorithm, use the following code:

```
PDFSign.HashAlgorithm = DigestAlgorithm.SHA256;
```

Also, the time stamping requests/responses could be made using other hash algorithms, as follow. Note that the time stamping server must support this algorithms.

```
PDFSign.TimeStamping.HashAlgorithm = DigestAlgorithm.SHA512;
```

Verifying the signing certificate

The signing certificate can be verified against OCSP and CRL before the signing operation.

```
PDFSign.VerifyDigitalCertificate(PDFSign.DigitalSignatureCertificate);
```

If you are behind a proxy, to get the OCSP or CRL response via web, set the proxy as follow:

```
System.Net.WebProxy proxy = new System.Net.WebProxy();  
proxy.Address = new Uri("http://myproxy.example.com:1024");  
proxy.Credentials = new System.Net.NetworkCredential("username", "password");  
PDFSign.ProxySettings = proxy;
```

Digital signature and encryption

If you want to protect the signed document by preventing actions like printing or content copying you must encrypt it. The document can be encrypted using passwords or digital certificates.

Password security

In order to encrypt the PDF document the *AppendSignature* property must be set to false. Also, the encryption algorithm must be specified using *EncryptionAlgorithm* property.

OwnerPassword property is used to set the password that protects the PDF document for printing or content copying.

To digitally sign and encrypt a PDF document using a password use the following code:

```
//Initializes a new instance of PDFSign class
PDFSign PDFSign = new PDFSign("Enter your application license here");

//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

//Load the certificate from .PFX
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopLeft;

//append signature must be set to false in order to encrypt de document
PDFSign.AppendSignature = false;

//set the document restrictions
PDFSign.Encryption.DocumentRestrictions =
PDFDocumentRestrictions.AllowContentCopying |
PDFDocumentRestrictions.AllowFillingOfFormFields;

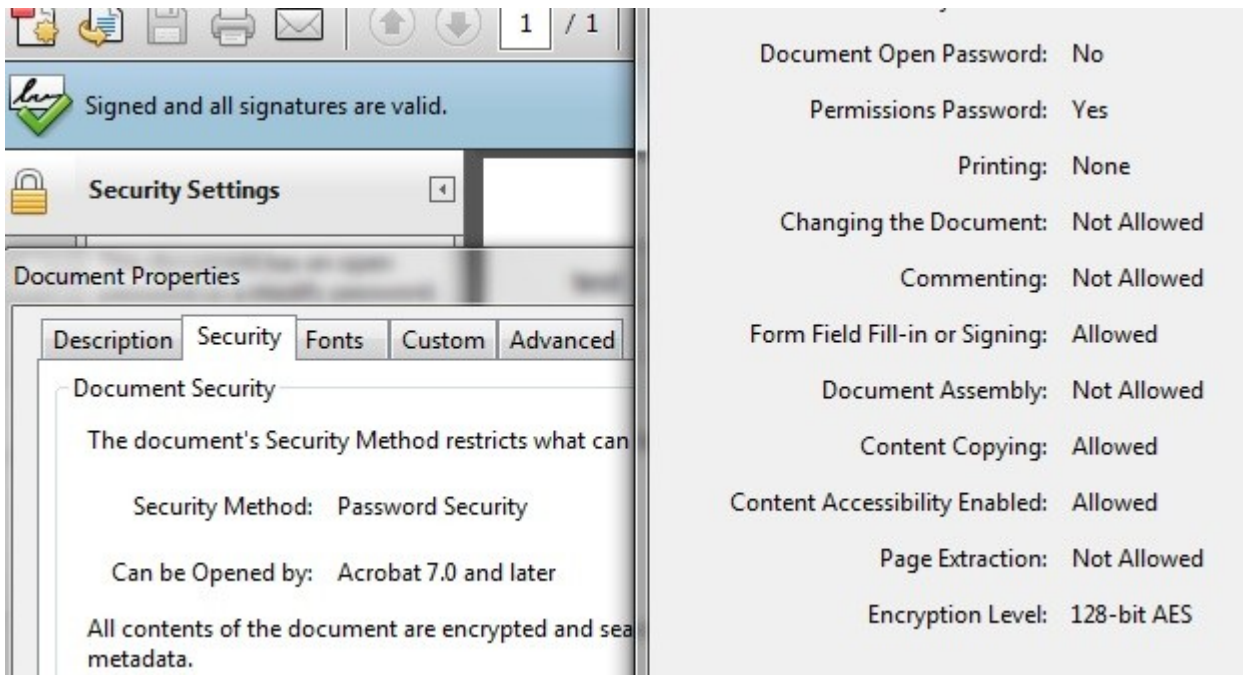
//set the encryption algorithm
PDFSign.Encryption.EncryptionAlgorithm =
PDFEncryptionAlgorithm.EnhancedEncryption128BitAES;

//set the encryption method
PDFSign.Encryption.EncryptionMethod = PDFEncryptionMethod.PasswordSecurity;

//set the owner password
PDFSign.Encryption.OwnerPassword = "123456";

//digitally sign, encrypt and save the PDF file
File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

When the signed and encrypted document is opened in a PDF reader, the security settings are shown like below.

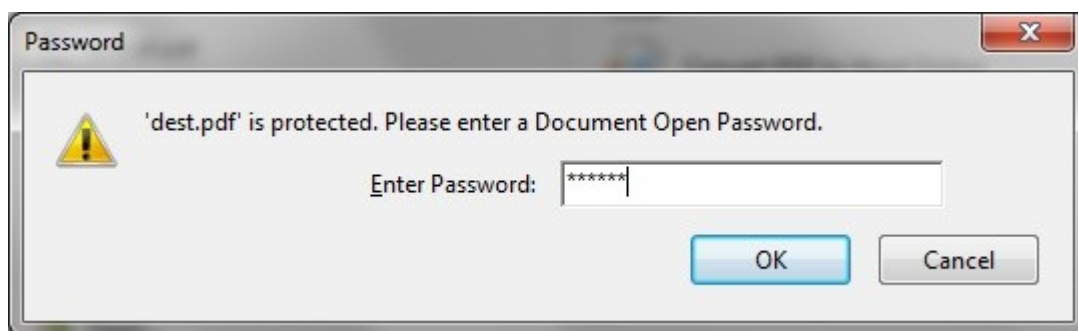


Security settings for a digitally sign and encrypted document

To digitally sign and protect the document with an opened password use the code below instead of the commented line:

```
//PDFSign.Encryption.OwnerPassword = "123456";  
PDFSign.Encryption.UserPassword = "123456";
```

When the document is opened in PDF reader, the password must be entered.



Password is required to open the document

Digital Certificate security

The document can be also protected using a digital certificate. Remember that the digital signature is created using the private key of the certificate. For the encryption the public key of the certificate is necessary. The public key of the encryption certificates are stored on *Microsoft Store – Other People* tab or in .cer files.

To encrypt a signed message using a digital certificate use the code below:

```
//append signature must be set to false in order to encrypt de document
PDFSign.AppendSignature = false;
//set the document restrictions
PDFSign.Encryption.DocumentRestrictions = PDFDocumentRestrictions.AllowNone;
//set the encryption algorithm
PDFSign.Encryption.EncryptionAlgorithm =
PDFEncryptionAlgorithm.StandardEncryption128BitRC4;
//set the encryption method
PDFSign.Encryption.EncryptionMethod = PDFEncryptionMethod.CertificateSecurity;

//use the digital certificates stored on Microsoft Store - Other People
PDFSign.Encryption.EncryptionCertificate = PDFSign.LoadCertificateFromStore(false,
"", "Encryption certificate", "Select the encryption certificate",
DigitalCertificateScope.ForEncryption);
```

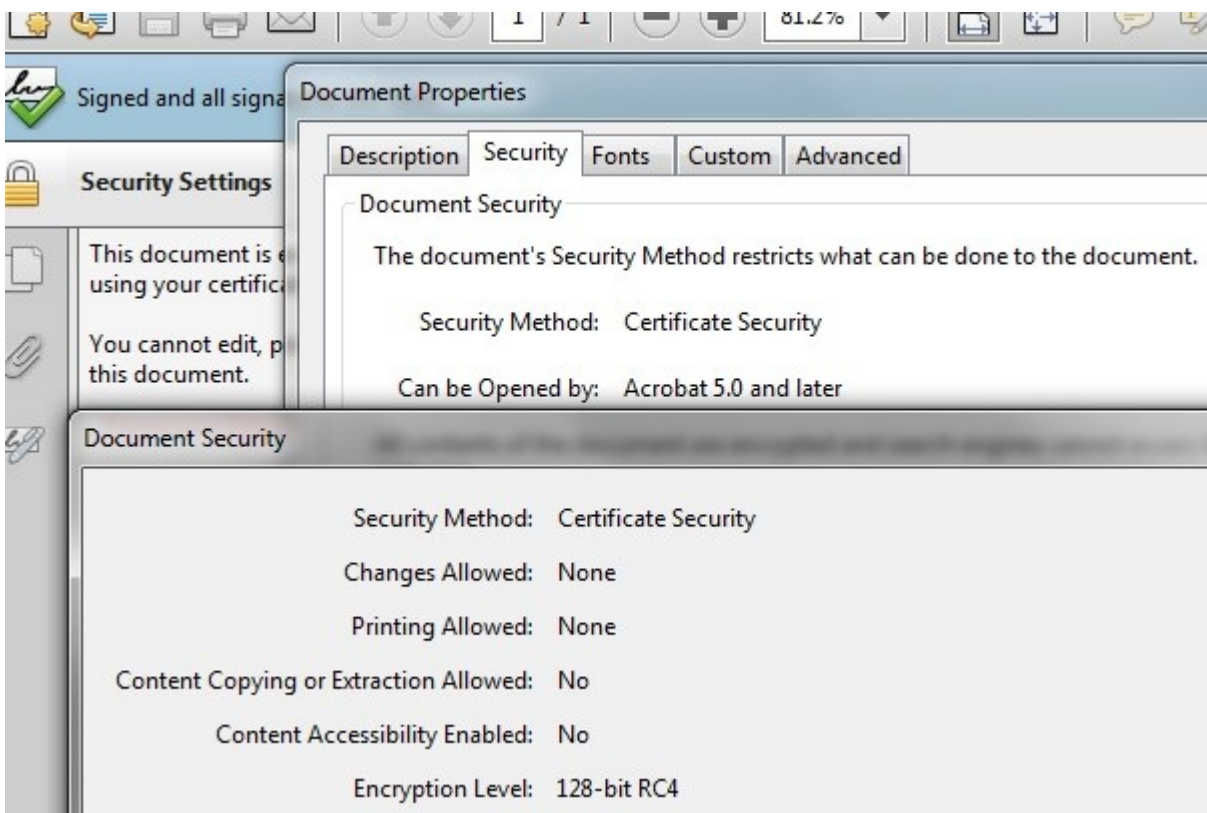
If you want to encrypt the PDF file using a .CER file (public key) use the code below instead of the commented lines:

```
//append signature must be set to false in order to encrypt de document
PDFSign.AppendSignature = false;
//set the document restrictions
PDFSign.Encryption.DocumentRestrictions = PDFDocumentRestrictions.AllowNone;
//set the encryption algorithm
PDFSign.Encryption.EncryptionAlgorithm =
PDFEncryptionAlgorithm.StandardEncryption128BitRC4;
//set the encryption method
PDFSign.Encryption.EncryptionMethod = PDFEncryptionMethod.CertificateSecurity;

//PDFSign.Encryption.EncryptionCertificate =
PDFSign.LoadCertificateFromStore(false, "", "Encryption certificate", "Select the
encryption certificate", DigitalCertificateScope.ForEncryption);

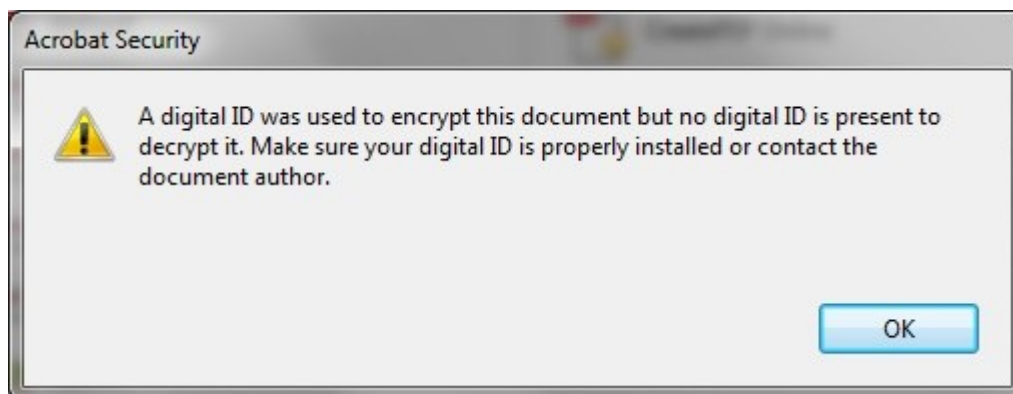
//loads the certificate from a .CER file
PDFSign.Encryption.EncryptionCertificate = new
System.Security.Cryptography.X509Certificates.X509Certificate2
(File.ReadAllBytes("c:\\encryption_certificate.cer"));
```

If the private key corresponding to the public key used for encryption is available on the computer where the the encrypted file is opened, the security settings are shown like below:



Security settings for a digitally sign and encrypted document

Observation: A file encrypted with the public key can be opened only by the corresponding private key of that certificate. If you want to encrypt a file for a person, you will need the public key of the certificate issued for that person. If the file is encrypted with your certificate only you can open that file. If the private key of the encryption certificate is not present a warning message will be displayed like below:



Decryption certificate (private key) is not present

Code snippets

Digitally sign all pages from a PDF file with a certificate stored on PFX file

```
PDFSign PDFSign = new PDFSign("Enter your application license here");
//load the pdf file
PDFSign.LoadPDFDocument("c:\\source.pdf");

//load the certificate
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
//put the signature to all pages
PDFSign.SignatureToAllPages = true;

//set the signature position
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopLeft;

//digitally sign and save the PDF file
File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

Set a custom signature rectangle and sign using a certificate from Microsoft Store

```
PDFSign PDFSign = new PDFSign("Enter your application license here");
PDFSign.LoadPDFDocument("c:\\source.pdf");

//load the certificate from Microsoft Store
PDFSign.DigitalSignatureCertificate = PDFSign.LoadCertificate(false, "", "Digital
certificates", "Select the digital certificate",
DigitalCertificateScope.ForDigitalSignature);

PDFSign.SignaturePage = 1;
//set the signature position
System.Drawing.Point pageRectangle = PDFSign.DocumentPageSize(1);

//put the signature on the middle of the page
PDFSign.SignaturePosition = new System.Drawing.Rectangle(pageRectangle.X / 2,
pageRectangle.Y / 2, 100, 50);

File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

Digitally sign a PDF located on the web only if it is not already signed

```
PDFSign PDFSign = new PDFSign("Enter your application license here");
//load the pdf file from web
PDFSign.LoadPDFDocument(new Uri("http://www.signfiles.com/test.pdf"));

//sign the document only if is not signed
if (PDFSign.DocumentProperties.NumberOfDigitalSignatures == 0)
{
    PDFSign.DigitalSignatureCertificate =
    PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
    PDFSign.SignaturePage = 1;
    PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopMiddle;
    File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
}
```

Set a custom text and font for the digital signature rectangle

```
PDFSign PDFSign = new PDFSign("Enter your application license here");

PDFSign.LoadPDFDocument("c:\\source.pdf");
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");

PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.BottomLeft;

//set some signature properties
PDFSign.SigningReason = "Signing reason";
PDFSign.SigningLocation = "Signing location";
PDFSign.SignerContactInformation = "Contact information";

//set the font file
PDFSign.FontFile = "c:\\windows\\fonts\\verdana.ttf";
//set the font size
PDFSign.FontSize = 6;

//customize the text that appears on the signature rectangle
PDFSign.SignatureText = "Signed by: " +
PDFSign.DigitalSignatureCertificate.GetNameInfo(X509NameType.SimpleName, false) +
"\nSigning time: " + DateTime.Now.ToShortDateString() +
"\nSigning reason: " + PDFSign.SigningReason +
"\nLocation: " + PDFSign.SigningLocation;

File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

Add an image on the signature rectangle and save the file as PDF/A

```
PDFSign PDFSign = new PDFSign("Enter your application license here");

PDFSign.LoadPDFDocument("c:\\source.pdf");
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");

PDFSign.SignaturePage = PDFSign.DocumentProperties.NumberOfPages;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.BottomRight;

PDFSign.SignatureText = "Signed by the author";

//path to the signature image
PDFSign.SignatureImage = File.ReadAllBytes("c:\\graphic.jpg");
PDFSign.SignatureImagePosition = SignatureImageType.ImageAsBackground;

//the font must be embedded in orde to save the file as PDF/A
PDFSign.FontFile = "c:\\windows\\fonts\\verdana.ttf";
PDFSign.SaveAsPDF/A = true;

File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

Set an invisible signature and certify a PDF file

```
PDFSign PDFSign = new PDFSign("Enter your application license here");
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");

PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopMiddle;

PDFSign.SigningLocation = "My location";
PDFSign.SigningReason = "I am the author of this document";

//certify the signature
PDFSign.CertifySignature = CertifyMethod.NoChangesAllowed;

//set an invisible signature
PDFSign.VisibleSignature = false;

File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

Time stamp a PDF file

```
PDFSign PDFSign = new PDFSign("Enter your application license here");
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");

PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopMiddle;

//simply set the TSA server URL
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");

File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

Time stamp a PDF file using authentication

```
PDFSign PDFSign = new PDFSign("Enter your application license here");
PDFSign.LoadPDFDocument("c:\\source.pdf");
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopMiddle;
PDFSign.OldStyleAdobeSignature = true;

//set the TSA server URL
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");
//set username and password
PDFSign.TimeStamping.UserName = "username";
PDFSign.TimeStamping.Password = "P@ssw0rd";

File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

Time stamp a PDF file using a proxy connection and set the TSA Policy OID

```
PDFSign PDFSign = new PDFSign("Enter your application license here");
PDFSign.LoadPDFDocument("c:\\source.pdf");
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopMiddle;

//set the TSA server URL
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");

//See:http://msdn.microsoft.com/en-us/library/system.net.httpwebrequest.proxy.aspx

//set the proxy settings
System.Net.WebProxy tsaProxy = new System.Net.WebProxy();
tsaProxy.Address = new Uri("http://myproxy.example.com:1024");
tsaProxy.Credentials = new System.Net.NetworkCredential("username", "password");

//use the proxy
PDFSign.ProxySettings = tsaProxy;

//set the TSA Policy OID
PDFSign.TimeStamping.PolicyOID = new System.Security.Cryptography.Oid("1.3.9.3");
File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```

Digitally sign and time stamp a folder with PDF files

```
PDFSign PDFSign = new PDFSign("enter your serial number");
PDFSign.LoadPDFDocument("c:\\source.pdf");
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopLeft;
PDFSign.SignaturePage = 1;

PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");

System.IO.DirectoryInfo di;
System.IO.FileInfo[] rgFiles;
//get the pdf files from the folder
di = new System.IO.DirectoryInfo("c:\\source_dir");
rgFiles = di.GetFiles("*.pdf");

foreach (FileInfo fi in rgFiles)
{
    //for readonly files
    fi.Attributes = FileAttributes.Normal;
    //load the PDF document
    PDFSign.LoadPDFDocumentFromFile(di.FullName + "\\\" + fi.Name);
    //digitally sign and save the PDF file
    File.WriteAllBytes("c:\\output_dir\\" + fi.Name,
PDFSign.ApplyDigitalSignature());
}
```

Digitally sign a PDF file in a ASP.NET application (IIS)

```
protected void Page_Load(object sender, EventArgs e)
{
    PDFSign PDFSign = new PDFSign("enter your serial number");

    //set the signing certificate
    //the PFX certificate must use MachineKeySet
    PDFSign.DigitalSignatureCertificate = new
System.Security.Cryptography.X509Certificates.X509Certificate2(Server.MapPath("cert.pfx"), "123456",
System.Security.Cryptography.X509Certificates.X509KeyStorageFlags.MachineKeySet);

    PDFSign.LoadPDFDocument(Server.MapPath("source.pdf"));

    PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopLeft;
    PDFSign.SignaturePage = 1;

    System.IO.File.WriteAllBytes(Server.MapPath("dest.pdf"),
PDFSign.ApplyDigitalSignature());
}
```

Verify a digital signature

In some cases is needed to verify the digital signatures attached to a PDF document.

To verify the digital signatures added to PDF document use the following code:

```
PDFSign PDFSign = new PDFSign("enter your serial number");
PDFSign.LoadPDFDocument("c:\\source.pdf");

foreach (PDFSignatureInformation psi in
PDFSign.DocumentProperties.DigitalSignatures)
{
    Console.WriteLine("Signature name: " + psi.SignatureName);
    Console.WriteLine("Digest Algorithm: " + psi.DigestAlgorithm);
    Console.WriteLine("Signing Reason: " + psi.SigningReason);
    Console.WriteLine("Signing location: " + psi.SigningLocation);
    Console.WriteLine("Signer: " + psi.DigitalSignatureCertificate.SubjectName.Name);
    Console.WriteLine("Signing date: " + psi.SignatureDate.ToString());
    Console.WriteLine("Signature is timestamped:" + psi.IsTimeStamped.ToString());
    Console.WriteLine("Is signature valid: " + psi.IsSignatureValid.ToString());
}
```

Merge multiple PDF files into a single PDF file

If you need to merge multiple PDF files into a single one, use the following code:

```
List<byte[]> sourceFiles = new List<byte[]>();

sourceFiles.Add(File.ReadAllBytes("c:\\1.pdf"));
sourceFiles.Add(File.ReadAllBytes("c:\\2.pdf"));
sourceFiles.Add(File.ReadAllBytes("c:\\3.pdf"));
sourceFiles.Add(File.ReadAllBytes("c:\\4.pdf"));

File.WriteAllBytes("c:\\merge.pdf", PdfMerge.MergeFiles(sourceFiles));
```

Insert texts and images in a PDF file

```
PdfInsertObject PdfInsertImage = new PdfInsertObject();

/*****
Insert images on PDF document
*****/
PdfInsertImage.LoadPDFDocument("c:\\source.pdf");

//adds an image on a specific rectangle location on the page 1. The image will be
placed over the PDF content of the page.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\watermark.png"), new
System.Drawing.Rectangle(10, 10, 100, 100), 1, ImagePosition.ImageOverContent);

//adds an image that will cover all the page 2. The image will be placed under the
PDF content (background) of the page.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\watermark.png"), 2,
ImagePosition.ImageUnderContent);

//adds an image that will start on a specific starting position on the page 3. The
image will not be resized. The image will be placed over the PDF content of the
page.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\watermark.png"), new
System.Drawing.Point(200, 200), 3, ImagePosition.ImageOverContent);

//adds an image on the top right corner of the document.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\signature_image.jpg"), new
System.Drawing.Rectangle(PdfInsertImage.DocumentProperties.DocumentPageSize(4).X -
100, PdfInsertImage.DocumentProperties.DocumentPageSize(4).Y - 100, 100, 100), 4,
ImagePosition.ImageOverContent);

//adds an image on the top left corner of the document.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\signature_image.jpg"), new
System.Drawing.Rectangle(0,
PdfInsertImage.DocumentProperties.DocumentPageSize(5).Y - 100, 100, 100), 5,
ImagePosition.ImageOverContent);

//adds an image on all document pages over the text.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\certificate_graphic.png"), new
System.Drawing.Point(100, 100), 0, ImagePosition.ImageOverContent);

//adds an image on all document pages under the text in the middle.
```

```

PdfInsertImage.AddImage(File.ReadAllBytes("c:\\watermark.png"), new
System.Drawing.Rectangle(PdfInsertImage.DocumentProperties.DocumentPageSize(3).X /
2, PdfInsertImage.DocumentProperties.DocumentPageSize(3).Y / 2, 100, 100), 0,
ImagePosition.ImageUnderContent);

/*****
Insert texts on PDF document
*****/

CustomText custText = new CustomText();
custText.Align = TextAlign.Left;
custText.FontFile = "c:\\arial.ttf";
custText.FontSize = 8;
custText.PageNumber = 1;
custText.StartingPointPosition = new System.Drawing.Point(100, 100);
custText.Text = "The first text inserted";
custText.TextColor = iTextSharp.text.BaseColor.BLUE;

PdfInsertImage.AddText(custText); //add the first text

CustomText custText2 = new CustomText();
custText2.Align = TextAlign.Left;
custText2.FontFile = "c:\\arial.ttf";
custText2.FontSize = 6;
custText2.PageNumber = 1;
custText2.StartingPointPosition = new System.Drawing.Point(80, 150);
custText2.TextDirection = TextDirection.RightToLeft;
custText2.Text = "ניהול קופה, ניהול מלאאי";
custText2.TextColor = iTextSharp.text.BaseColor.BLACK;

PdfInsertImage.AddText(custText2); //add the second text

File.WriteAllBytes("c:\\destination.pdf", PdfInsertImage.InsertObjects());
//insert objects and save the PDF file

```

Exceptions

All general exceptions strings thrown by the library are available on the class *ExceptionsStrings*. All other exceptions are thrown by the PDF engine or by the .NET Framework.

The PDFSignDLL methods throw the following exceptions:

```
PDFSign(string librarySerialNumberLicense)
throw new ArgumentException("Licensing error: " + ex.Message);
```

```
X509Certificate2 LoadCertificate(byte[] certificate, string password)
//"CryptographicException": Throws an exception if the certificate cannot be
loaded
throw;
```

```
X509Certificate2 LoadCertificate(bool validOnly, string issuerName, string
windowTitle, string windowDescription, DigitalCertificateScope certificateScope)
//"InvalidOperationException": Throws an exception if the certificate is invalid
//"CryptographicException": Throws an exception if the certificate cannot be
loaded
throw;
```

```
X509Certificate2 LoadCertificate(bool validOnly, DigitalCertificateSearchCriteria
selectionType, string searchString, DigitalCertificateScope certificateScope)
//"FormatException": Throws an exception if the criteria in incorrect
//"CryptographicException": Throws an exception if the certificate cannot be
loaded.
throw;
```

```
byte[] ApplyDigitalSignature()

if (reader == null)
    throw new NullReferenceException(ExceptionsStrings.DocumentIsNotLoaded);

if (DigitalSignatureCertificate == null)
    throw new
NullReferenceException(ExceptionsStrings.DigitalCertificateIsNotSet);

//certificate encryption
if (Encryption.EncryptionMethod == PDFEncryptionMethod.CertificateSecurity)
{
    if (Encryption.EncryptionCertificate == null)
        throw new
NullReferenceException(ExceptionsStrings.DigitalCertificateIsNotSet);
}
if (File.Exists(FontFile) == false)
    throw new System.IO.FileNotFoundException();

if (SignaturePage < 1)
    //throw new Exception("Invalid signature page number
    throw new
System.ArgumentOutOfRangeException(ExceptionsStrings.InvalidPageNumber);
```

```

if (SignaturePage > reader.NumberOfPages)
    throw new
System.ArgumentOutOfRangeException(ExceptionsStrings.InvalidPageNumber);

System.Drawing.Point DocumentPageSize(int pageNumber)
//"ArgumentOutOfRangeException": Throws an exception if the PDF document is not
loaded or the document page is invalid.

//"Exception": Throws an exception if the PDF document is not loaded or the
document page is invalid.

if (reader == null)
    throw new
ArgumentOutOfRangeException(ExceptionsStrings.DocumentIsNotLoaded);

if (pageNumber > DocumentProperties.NumberOfPages || pageNumber <= 0)
    throw new ArgumentOutOfRangeException();
throw;

LoadPDFDocument(byte[] pdfArray)
//"Exception": Throws an exception if an error occurred

throw;

void LoadPDFDocument(string PdfFile)

if (File.Exists(PdfFile) == false)
    throw new System.IO.FileNotFoundException();

throw;

```

```

void LoadPDFDocumen(Uri PdfUrl)
throw;

DigitalCertificateStatus VerifyDigitalCertificate(X509Certificate2 certificate)
...
catch (Exception ex)
{
    throw new System.Security.Cryptography.CryptographicException("Certificate
validation error: " + ex.Message);
}

```

```

private byte[] GetTimestampToken()
throw new System.Net.WebException(ExceptionsStrings.InvalidTimeStampingResponse);

```

License Agreement (EULA)

Important! Read the following terms carefully before installing, copying and/or using the product. Installing, copying or using the product indicates your acceptance of these terms, as well the terms in the contract between Client and Secure Soft.

This End-User License Agreement ("EULA") is a legal agreement between Client and Secure Soft governing the use of the software (SOFTWARE) accompanying this EULA, including any and all associated media, printed materials, and "online" or electronic documentation protected by copyright laws. By installing, copying, or otherwise using the SOFTWARE, you agree to be bound by the terms of this EULA. If you do not agree with the terms of this EULA, do not use the SOFTWARE.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall Secure Soft or its suppliers be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use the SOFTWARE, even if Secure Soft has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability, for consequential or incidental damages, the above limitation may not apply to you. Liability of the Vendor will be limited to a maximum of the original purchase price of the Software. The Vendor will not be liable for any general, special, incidental or consequential damages including, but not limited to, loss of production, loss of profits, loss of revenue, loss of data, or any other business or economic disadvantage suffered by the Licensee arising out of the use or failure to use the Software. The Vendor makes no warranty expressed or implied regarding the fitness of the Software for a particular purpose or that the Software will be suitable or appropriate for the specific requirements of the Licensee. The Vendor does not warrant that use of the Software will be uninterrupted or error-free. The Licensee accepts that software in general is prone to bugs and flaws within an acceptable level as determined in the industry.

TERMINATION. Without prejudice to any other rights, Secure Soft may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such an event, you must destroy all copies of the SOFTWARE, all of its component parts and uninstall the SOFTWARE.

WARRANTIES. The SOFTWARE is supplied "as is", Secure Soft does not guarantee that the SOFTWARE will carry no errors, nor will it take on any liability for damages more than written here. Secure Soft does not warrant that the SOFTWARE will meet further requirements. However, Secure Soft states that every reasonable effort has been made to avoid presence of any possible malevolent code in the SOFTWARE (including, but not limited to, viruses, trojans, root kits and/or spyware) and declares that to the best knowledge of competent experts of Secure Soft the SOFTWARE does not contain any such malware. Supplemental enhances and updates are subject to other contracts.

NO OTHER WARRANTIES. Secure Soft disclaims all other warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a

particular purpose, with respect to the SOFTWARE and the accompanying written materials. This limited warranty gives you specific legal rights. You may have other which vary from state to state.

MISCELLANEOUS. This EULA comes into force when you accept all the conditions stated herein.

CONTROLLING LAW AND SEVERABILITY. This License shall be governed by the laws of the Romania. If for any reason a court of competent jurisdiction finds any provision, or portion thereof, to be unenforceable, the remainder of this License shall continue in full force and effect.

COPYRIGHT NOTES

Copyright (c) 2000 - 2007 The Legion Of The Bouncy Castle (<http://www.bouncycastle.org>)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

This product uses iText library (<http://sourceforge.net/projects/itextsharp/>). All changes made to the iTextsharp library (the source code) are available at this link:

<http://www.signfiles.com/license/LicenseAgreement.zip>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU Affero General Public License version 3 as published by the Free Software Foundation with the addition of the following permission added to Section 15 as permitted in Section 7(a):
FOR ANY PART OF THE COVERED WORK IN WHICH THE COPYRIGHT IS OWNED BY 1T3XT, 1T3XT DISCLAIMS THE WARRANTY OF NON INFRINGEMENT OF THIRD PARTY RIGHTS.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program; if not, see <http://www.gnu.org/licenses> or write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA, 02110-1301 USA, or download the license from the following URL:

<http://itextpdf.com/terms-of-use/>

The interactive user interfaces in modified source and object code versions of this program

must display Appropriate Legal Notices, as required under Section 5 of the GNU Affero General Public License.

In accordance with Section 7(b) of the GNU Affero General Public License, you must retain the producer line in every PDF that is created or manipulated using iText.

You can be released from the requirements of the license by purchasing a commercial license. Buying such a license is mandatory as soon as you develop commercial activities involving the iText software without disclosing the source code of your own applications. These activities include: offering paid services to customers as an ASP, serving PDFs on the fly in a web application, shipping iText with a closed source product.

For more information, please contact iText Software Corp. at this address:
sales@itextpdf.com

All other trademarks are property of their respective owners.