DigiDoc COM Programmer’s Guide

Document Version: 3.7
Library Version: 3.7
Last update: 22.01.2013

1. Document versions

Document information	
Created on	22.01.2013
Reference	DigiDoc COM Programmer's Guide
Receiver	Sertifitseerimiskeskus AS
Author	Veiko Sinivee, Kersti Üts, Kristi Uukkivi
Version	3.7

Version information		
Date	**Version**	**Changes**
	2.2.5.2	The latest version of "DigiDocLibCOM.DLL user manual" created by Veiko Sinivee
22.01.2013	3.7	Document updated to 3.7 version

Table of contents

2. Introduction

This document describes DigiDoc COM (Component Object Model) software library which is a part of the OpenXadES/DigiDoc system. It is a language independent binary object model which is based on the CDigiDoc library of the DigiDoc system.

CDigiDoc is a software library in C programming language, it is a basic building tool for creating applications which handle digital signatures, encrypt and decrypt files. DigiDoc COM library can be used as an intermediate layer for integrating CDigiDoc library's digital signature handling functionality with client applications written for Windows environment.

The digitally signed files are created in "DigiDoc format" (with .ddoc file extension), compliant to XML Advanced Electronic Signatures (XAdES), technical standard published by European Telecommunication Standards Institute (ETSI).

This document covers the following information about DigiDoc COM library:

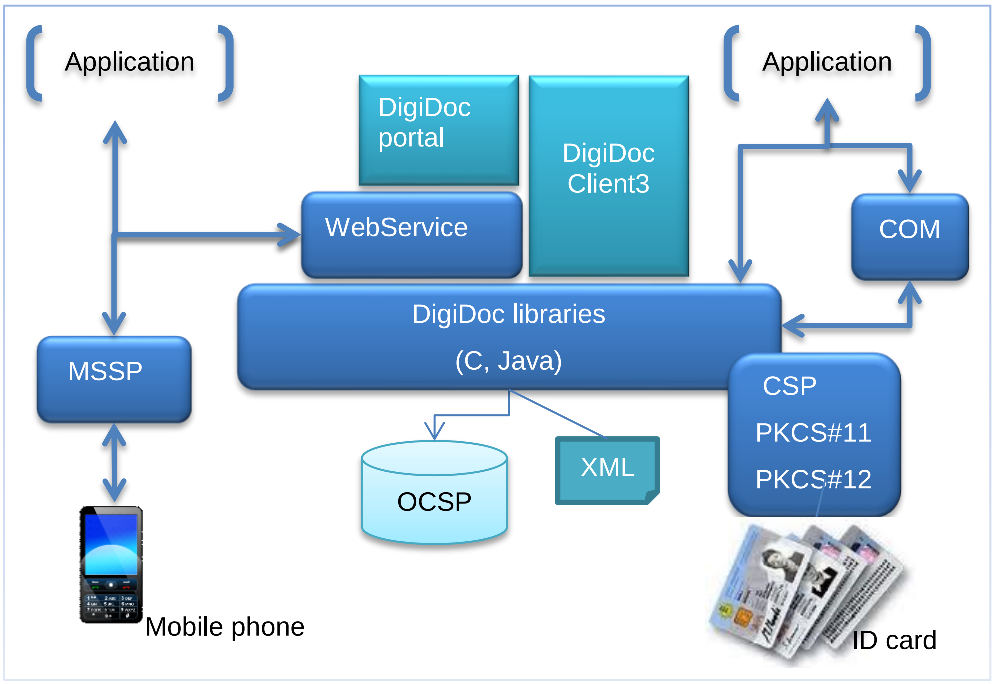
- Section 2 introduces the OpenXAdES/DigiDoc framework, its general security model and formats available for digitally signed files.
- Section 3 gives an overview of the system requirements and describes API's usage for some of the most commonly used document signing tasks.
- Section 4 explains using the command line utility program for DigiDoc COM, including sample use cases.
- Appendix 1 provides a description of DigiDoc COM's classes and their components

CDigiDoc library of DigiDoc system is described in a separate document, see [12].

For implementing file encryption and decryption functions in applications running on Windows environment, please refer to the NDigiDoc library [14]. NDigiDoc is a .NET library for digital encryption and decryption of files according to the XML-ENC standard.

2.1. About DigiDoc

DigiDoc COM library forms a part of the wider OpenXAdES/DigiDoc system framework which offers a full-scale architecture for digital signature and documents, consisting of software libraries (C and Java), web service and end-user applications such as DigiDoc Portal and DigiDoc Client3 according to the following figure:



1 DigiDoc framework

It is easy to integrate DigiDoc components into existing applications in order to allow for creation, handling, forwarding and verification of digital signatures and support file encryption/decryption. All applications share a common digitally signed file format (current version DIGIDOC-XML 1.3) which is a profile of XAdES.

2.2. DigiDoc security model

The general security model of the DigiDoc and OpenXAdES ideology works by obtaining proof of validity of the signer's X.509 digital certificate issued by a certificate authority (CA) at the time of signature creation.

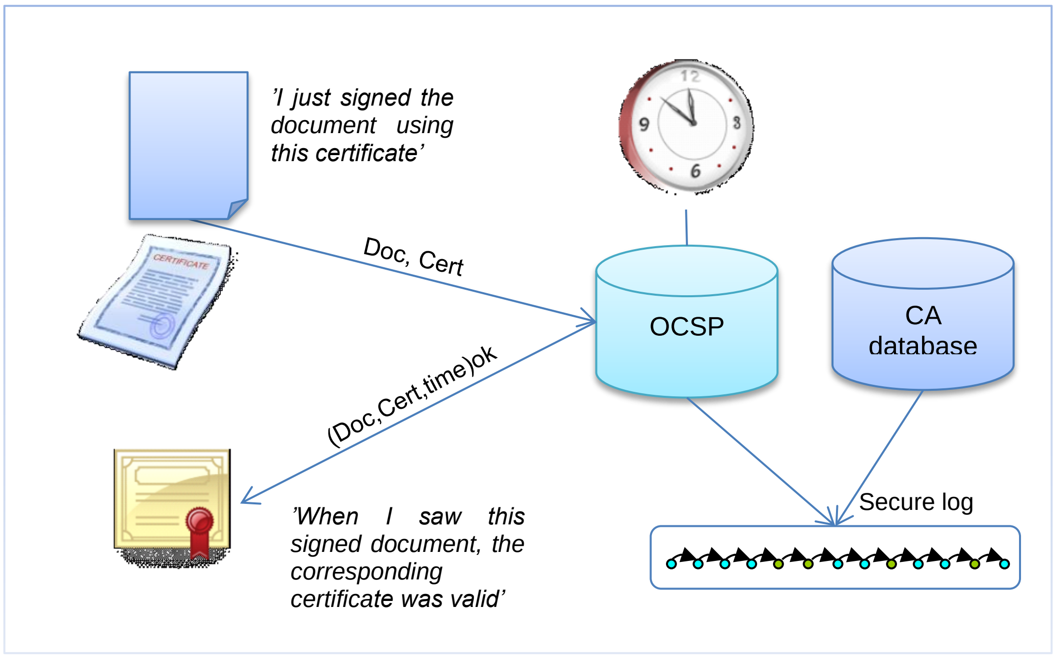
This proof is obtained in the format of Online Certificate Status Protocol (OCSP) response and stored within the signed document. Furthermore, (hash of the) created signature is sent within the OCSP request and received back within the response. This allows interpreting of the positive OCSP response as "at the time I saw this digitally signed file, corresponding certificate was valid".

The OCSP service is acting as a digital e-notary confirming signatures created locally with a smart card. From infrastructure side, this security model requires a standard OCSP responder. Hash of the signature is placed on the "nonce" field of the OCSP request structure. In order to achieve the freshest certificate validity information, it is recommended to run the OCSP responder in "real-time" mode meaning that:

- certificate validity information is obtained from live database rather than from CRL (Certificate Revocation List)
- the time value in the OCSP response is actual (as precise as possible)

To achieve long-time validity of digital signatures, a secure log system is employed within the model. All OCSP responses and changes in certificate validity are securely logged to preserve digital signature validity even after private key compromise of CA or OCSP responder. It is important to notice that additional time-stamps are not necessary when employing the security model described:

- time of signing and time of obtaining validity information is indicated in the OCSP response
- the secure log provides for long-time validity without need for archival timestamps


'I just signed the document using this certificate'
Doc, Cert
OCSP
CA database
(Doc,Cert,time)ok
'When I saw this signed document, the corresponding certificate was valid'
Secure log

2 DigiDoc security model

2.3. Format of digitally signed file

The format of the digitally signed file is based on **ETSI TS 101 903** standard called **XML Advanced Electronic Signatures (XAdES)**. This standard provides syntax for digital signatures with various levels of additional validity information. DigiDoc COM is based on the CDigiDoc library which is implementing a subset of these standards.

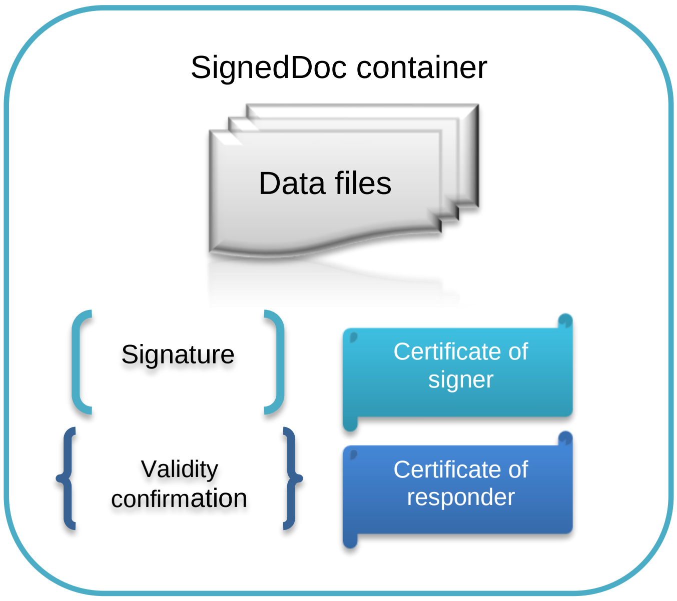
In order to comply with the security model described above, the XAdES profile **XAdES-X-L** is used in the DigiDoc system but "**time-marks**" are used instead of "time-stamps" – signing (and certificate validation) time comes with OCSP response.

This profile:

- allows for incorporating following signed properties
  - Certificate used for signing
  - Signing time
  - Signature production place
  - Signer role or resolution
- incorporates full certificate validity information within the signature
  - OCSP response

- OCSP responder certificate

As a result, it is possible to verify signature validity without any additional external information – the verifier should trust the issuer of signer's certificate and the OCSP responder's certificate. Original files (which were signed) along with the signature(s), validation confirmation(s) and certificates are encapsulated within container with "SignedDoc" as a root element.



3 SignedDoc container

The library currently offers **DIGIDOC-XML** document format to be used.

The DIGIDOC-XML document format (latest version 1.3) is fully conforming to XAdES standard (note however that not every single detail allowed in XAdES standard is supported).

DigiDoc system uses file extension **.ddoc** to distinguish digitally signed files according to the described file format. Syntax of the .ddoc file is described in a separate document in detail (see [6]).

The DIGIDOC-XML document's container is a single XML file which may contain embedded data file(s) and signature(s). It is possible to add data files to the container by:

- embedding binary data in base64 encoding (CONTENT_EMBEDDED_BASE64 mode),
- embedding pure text or XML – no longer supported (CONTENT_EMBEDDED mode),
- adding only reference to an external file – no longer supported (DETACHED mode).

SHA-1 digest type is supported and set automatically.

3. Overview

The following section describes the DigiDoc COM library's principles, dependencies and usage possibilities in client applications by demonstrating the most common API calls for digital signature creation.

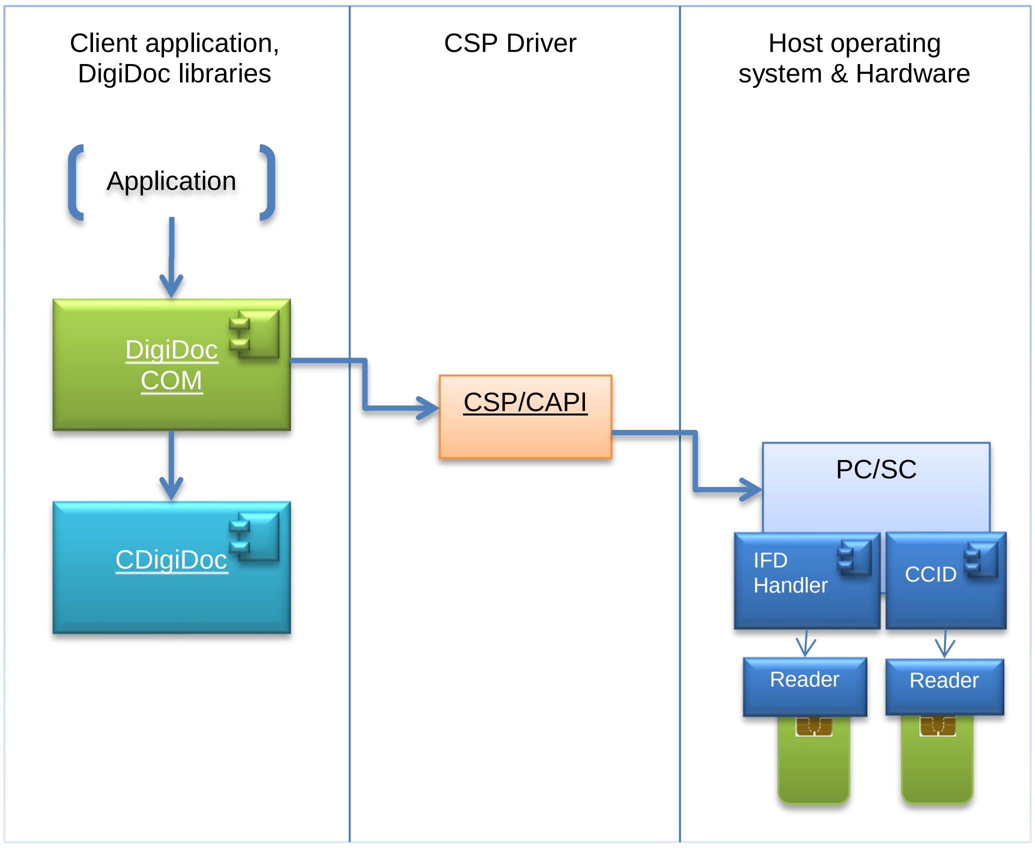
DigiDoc COM is a binary object model, based on the CDigiDoc library which has been linked into COM DLL from the source code.

DigiDoc COM library offers the possibility to integrate the following CDigiDoc's functionality to client applications running on 32-bit Windows platform:

- Creating files in supported DigiDoc formats (current default in **bold**):
  - **DIGIDOC-XML 1.3**
- Digitally **signing** the DigiDoc files using smart cards
- Adding **time marks** and **validity confirmations** to digital signatures using OCSP protocol.
- **Verifying** the digital signatures.

Note: older DigiDoc file formats SK-XML, DIGIDOC-XML 1.1 and DIGIDOC-XML 1.2 are supported only for backward compatibility in case of digital signature verification and data file extraction operations (creating new files and adding signatures is no longer supported).

Digital signing is implemented with Cryptographic Service Provider (CSP) over Microsoft CryptoAPI.



4 Sample DigiDoc COM implementation using smart cards for digital signing

Component	Description
CSP	Cryptographic Service Provider. Used for enabling cryptographic functions with smart cards in Windows environment.
CAPI	Microsoft CryptoAPI. A set of dynamically linked libraries which provide an interface for CSP calls.
PC/SC	Standard communication interface between the computer and the smart card, a cross-platform API for accessing smart card readers
IFDHandler	Interface Device Handler for CCID readers
CCID	USB driver for Chip/Smart Card Interface Devices
Reader	Device used for communication with a smart card

The COM layer is as thin as possible, meaning that it does not replace the CDigiDoc's structures but holds their references, organizes memory management and forwards function calls.

DigiDoc COM uses the CDigiDoc library's configuration file for compatibility purposes. Configuration data is retrieved from the CDigiDoc's configuration file 'digidoc.ini' during DigiDoc COM library's initialization.

Certificates to be used during signature creation and OCSP confirmation adding are retrieved from Windows Certificate Store. Thus, the following certificates have to be installed in the Certificate Store prior to conducting the mentioned operations:

- certificate of the CA of the signer of the document,
- certificate of the OCSP responder which is used to get the validity confirmation.

Note that no additional actions are needed to be taken when using real-life Estonian ID cards for signing as the Estonian CA's live certificates are included in the DigiDoc COM library's distribution package and are installed in the Windows Certificate Store during the library's installation process.

In order to use test ID cards for signature creation, you need to install test certificates separately (the installation package is accessible from http://id.ee/?id=28735).

Note: test certificates should not be used in live applications as the DigiDoc COM library does not give notifications to the user in case of test signatures.

3.1. References and additional resources

[1] RFC2560	Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C., X.509 Internet Public Key Infrastructure: Online Certificate Status Protocol - OCSP. June 1999
[2] RFC3275	Eastlake 3rd D., Reagle J., Solo D., (Extensible Markup Language) XML Signature Syntax and Processing. (XML-DSIG) March 2002.
[3] ETSI TS 101 903	XML Advanced Electronic Signatures (XAdES). February 2002
[4] XML Schema 2	XML Schema Part 2: Data types. W3C Recommendation 02 May 2001

	(http://www.w3.org/TR/xmlschema-2/)
[5] DSA	Estonian Digital Signature Act
[6] DigiDoc format	DigiDoc file format (http://www.id.ee/public/DigiDoci_vorming_1.3.2.pdf)
[7] XML-ENC	http://www.w3.org/TR/xmlenc-core/
[8] DigiDocService Specification	EN: http://sk.ee/upload/files/DigiDocService_spec_eng.pdf ET: http://www.sk.ee/upload/files/DigiDocService_spec_est.pdf
[9] ESTEID	ESTEID Card Certification Policy (http://sk.ee/upload/files/SK-CP-ESTEID-3_2_en.pdf) Certificates on identity cards of Republic of Estonia (http://sk.ee/upload/files/ESTEID_profiil_en-3_3.pdf)
[10] CSP (MS CSP)	Microsoft Cryptographic Service Provider
[11] PKCS#11	RSA Laboratories Cryptographic Token Interface Standard
[12] CDigiDoc library	CDigiDoc programmer’s guide (http://id.ee/index.php?id=35782)
[13] Release notes	DigiDoc COM library’s release notes (http://id.ee/index.php?id=35784)
[14] NDigiDoc library	NDigiDoc Programmer’s Guide (http://www.id.ee/index.php?id=35857)

3.2. *Terms and acronyms*

CRL	Certificate Revocation List, a list of certificates (or more specifically, a list of serial numbers for certificates) that have been revoked, and therefore should not be relied upon.
DIGIDOC-XML (.ddoc)	The term is used to denote a DigiDoc document format that is based on the XAdES standard and is a profile of that standard. The profile does not exactly match any subsets described in XAdES standard – the best format name would be “XAdES-C-L” indicating that all certificates and OCSP confirmations are present but there are no “pure” timestamps. A DIGIDOC-XML file is basically a <SignedDoc /> container that contains original data files and signatures. The file extension for DIGIDOC-XML file format is “.ddoc”, MIME-type is “application/ddoc”.
OCSP	Online Certificate Status Protocol, an Internet protocol used for obtaining the revocation status of an X.509 digital certificate
OCSP Responder	OCSP Server, maintains a store of CA-published CRLs and an up-to-date list of valid and invalid certificates. After the OCSP responder receives a validation request (typically an HTTP or HTTPS transmission), the OCSP

	responder either validates the status of the certificate using its own authentication database or calls upon the OCSP responder that originally issued the certificate to validate the request. After formulating a response, the OCSP responder returns the signed response, and the original certificate is either approved or rejected, based on whether or not the OCSP responder validates the certificate.
X.509	an ITU-T standard for a public key infrastructure (PKI) and Privilege Management Infrastructure (PMI) which specifies standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm
XAdES	XML Advanced Electronic Signatures, a set of extensions to XML-DSig recommendation making it suitable for advanced electronic signature. Specifies precise profiles of XML-DSig for use with advanced electronic signature in the meaning of European Union Directive 1999/93/EC.
XML-DSig	a general framework for digitally signing documents, defines an XML syntax for digital signatures and is defined in the W3C recommendation XML Signature Syntax and Processing

3.3. Dependencies

DigiDoc COM depends on the CDigiDoc library and its components. See the CDigiDoc documentation (section 3.3 Dependencies) for more information [12].

Note that all of the components which are necessary for DigiDoc COM library's functioning are included in the library's installation package.

3.4. Digital signing by using API

DigiDoc COM library enables creating, signing and verification of digitally signed documents, according to XAdES (ETSI TS101903) and XML-DSIG standards. In the next chapters a short introduction is given on the main function calls that can be made by a client application to accomplish the above mentioned. The client application in the examples has been implemented in C++ programming language.

In general, the procedure of creating and signing a new DigiDoc document with DigiDoc COM library consists of the following steps (with the used COM object given in brackets):

- initialize a new DigiDoc document (ComSignedDoc),
- add data file(s) to the document (ComDataFile),
- add signature(s) (ComSignatureInfo),
- add OCSP confirmation to the created signature(s),
- write the newly created DigiDoc document to a file.

For a complementary description of the library's components referenced in the following chapters, see Appendix 1.

3.4.1. Initialization

Call the following function in order to initialize the DigiDoc COM library:

```
HRESULT hr = CoInitialize(NULL);
```

Create the ComDigiDocLib object:

```
CLSID g_clsid; // a COM class object identifier
// Get the class identifier for ComDigiDocLib
hr = CLSIDFromProgID(OLESTR("DigiDocLibCOM.ComDigiDocLib"), &g_clsid);
// Create instance of the class
hr = CoCreateInstance(g_clsid, NULL, CLSCTX_INPROC_SERVER,
      __uuidof(IComDigiDocLib), (LPVOID *)&g_pLib);
```

ComDigiDocLib object contains general methods which are not directly bound to any particular COM class.

During DigiDoc COM library's initialization, configuration data is retrieved from the CDigiDoc's configuration file 'digidoc.ini'.

3.4.2. Creating a DigiDoc document

Firstly, define pointer to the required interface:

```
IComSignedDoc *g_pSigDoc = NULL; // interface for ComSignedDoc
```

Create an instance of ComSignedDoc object:

```
// Get the class identifier for ComDigiDocLib
hr = CLSIDFromProgID(OLESTR("DigiDocLibCOM.ComSignedDoc"), &g_clsid);
// Create instance of the class
hr = CoCreateInstance(g_clsid, NULL, CLSCTX_INPROC_SERVER,
        __uuidof(IComSignedDoc), (LPVOID *)&g_pSigDoc);
```

The ComSignedDoc object reflects the file format of DigiDoc. All other relevant objects are part of this basic structure.

Next, initialize the ComSignedDoc object:

```
g_pSigDoc->initialize("DIGIDOC-XML", // format of the document
      "1.3"); // default version number
```

Note: the functionality of creating new files in older DigiDoc file formats SK-XML, DIGIDOC-XML 1.1 and DIGIDOC-XML 1.2 is no longer supported.

In the following sections, we add a data file and a signature to the DigiDoc structure before writing it into an output file.

3.4.3. Adding data files

One ComDataFile object corresponds to one original data file (file-to-be-signed) in DigiDoc container. One DigiDoc container can incorporate multiple data files. The data files are embedded in the DigiDoc container.

Create an instance of ComDataFile:

```
IComDataFile* pDf = NULL; // interface for ComDataFile
// Create instance of ComDataFile
hr = CLSIDFromProgID(OLESTR("DigiDocLibCOM.ComDataFile"), &g_clsid);
hr = CoCreateInstance(g_clsid, NULL, CLSCTX_INPROC_SERVER,
      __uuidof(IComDataFile), (LPVOID *)&pDf);
```

You can add a data file to a DigiDoc container by calling the function IComSignedDoc.createDataFile(). The function creates a new DataFile element and saves the original data file in DigiDoc container:

```
// Define the data file's attributes as BSTR type variables

BSTR fname = _com_util::ConvertStringToBSTR("<data file's name>");
BSTR content = _com_util::ConvertStringToBSTR("<file embedding option>");
BSTR mime = _com_util::ConvertStringToBSTR("<mime-type>");
BSTR dType = _com_util::ConvertStringToBSTR("");
BSTR charset = _com_util::ConvertStringToBSTR("UTF-8");
V_VT(&vDig)= VT_EMPTY;

// Add the data file to DigiDoc container

hr = g_pSigDoc->createDataFile(fname, // data file name
      content,       // data file's embedding option
      mime,          // mime type
      0,             // file size in bytes
      vDig,          // data file's hash value
      0,             // hash length
      dType,         // type of hash algorithm
      charset,       // data file's charset. UTF-8 encoding should be used
      pDf);  // pointer to the data file
```

The second parameter of the function above reflects how data files are embedded in the DigiDoc container. Supported embedding option is "`CONTENT_EMBEDDED_BASE64`" – contents of the data file are encoded using base64-encoding before merging it into DigiDoc container.

Third parameter is a MIME type of the data file. For example "text/plain", "application/msword" or "application/pdf", depending on the type of the data file.

In most cases, the next four parameters should be left to the library to determine. The parameters specify:

- size of the data file in bytes,
- hash of the data file,
- size of the hash of the data file,
- type of hash algorithm (only SHA-1 is supported).

To calculate the values of these four parameters, do as follows:

```
// Get the data file's identifier as BSTR

BSTR bstr;
hr = pDf->get_szId(&bstr);
BSTR id = _com_util::ConvertBSTRToString(bstr);

// Calculate the parameter values

hr = g_pSigDoc->calculateDataFileSizeAndDigest(id, // data file's id
      fname, // data file's name
      0); // hash type. SHA-1 will be used automatically

// Finally, release the interface pointer

pDf->Release();
```

3.4.4. Adding signatures

You can sign with a smart card (e.g. Estonian ID card).

Note: the functionality of adding signatures is no longer supported in case of older DigiDoc file formats SK-XML, DIGIDOC-XML 1.1 and DIGIDOC-XML 1.2.

ComSignatureInfo object is needed to incorporate the necessary information about the signature before it can be created. Create an instance of the object as follows:

```
IComSignatureInfo* pSig = NULL;

hr = CLSIDFromProgID(OLESTR("DigiDocLibCOM.ComSignatureInfo"), &g_clsid);
hr = CoCreateInstance(g_clsid, NULL, CLSCTX_INPROC_SERVER,
      __uuidof(IComSignatureInfo), (LPVOID *)&pSig);

hr = g_pSigDoc->createSignatureInfo(pSig);
```

The IComSignedDoc.createSignatureInfo() function creates an 'empty' signature to which any relevant data can now be added. Call the function IComSignedDoc.addAllDocInfos() to add all of the DigiDoc container's data files to the signature object:

```
g_pSigDoc->addAllDocInfos(pSig);
```

Optionally, you can add meta-data of the signer by using the following functions:

```
IComSignatureInfo.addSignerRole(VARIANT tfCertified, BSTR bRole, VARIANT
tfEncode);
IComSignatureInfo.setSignatureProductionPlace(BSTR bCity, BSTR bState,
      BSTR bZip, BSTR bCountry);
```

Calculate the signature as follows:

```
long lErr = 0;
pSig->calculateSignatureWithCSPEstID(g_pSigDoc, // ComSignedDoc object
      0,
      &lErr); // return error code
```

Finally, release the interface pointer:

```
pSig->Release();
```

3.4.5. Adding an OCSP confirmation

OCSP protocol is used to get validity confirmation from OCSP responder to prove that certificate was valid at the time of signing.

You can add the confirmation by calling out the IComSignedDoc.getConfirmation() function:

```
// Define the attributes as BSTR type variables

BSTR bPkcs12 = _com_util::ConvertStringToBSTR("<pkcs12 file name>");
BSTR bP12Passwd = _com_util::ConvertStringToBSTR("<pkcs12 container
      password>");
BSTR bUrl = _com_util::ConvertStringToBSTR("<OCSP responder URL>");
BSTR bProxyHost = _com_util::ConvertStringToBSTR("<proxy hostname>");
BSTR bProxyPort = _com_util::ConvertStringToBSTR("<proxy port>");

g_pSigDoc->getConfirmation(pSig, // ComSignatureInfo object
      bPkcs12,       // pkcs12 file name
      bP12Passwd,    // pkcs12 container's password
      bUrl,          // OCSP responder's URL
      bProxyHost,    // proxy host if using a proxy
      bProxyPort,    // proxy port
      &lErr);        // return code
```

If parameters bPkcs12, bP12Passwd, bUrl, bProxyHost and bProxyPort are set to NULL then the respective values and their usage rules are looked up from the configuration file's entries (see the CDigiDoc library's documentation (section 3.4 Configuring CDigiDoc) for more information [12]).

3.4.6. Reading and writing DigiDoc documents

For creating a file in DigiDoc format do as follows:

```
BSTR oldfile = _com_util::ConvertStringToBSTR("<existing ddoc file
      name>");
BSTR newfile = _com_util::ConvertStringToBSTR("<new ddoc file name>");
long lErr = 0;

hr = g_pSigDoc->createSignedDoc(newfile, // new ddoc output file's name
      oldfile, // specifies existing ddoc file, if necessary
      &lErr); // return code
```

The "oldfile" parameter value can be set to NULL if you are creating a new DigiDoc document from scratch. If you have previously read in an existing DigiDoc document to modify it (e.g. add signature(s) or data file(s)) and now try to write it to an output file then you have to specify the existing DigiDoc file's path and filename in the "oldfile" parameter. Otherwise the data file contents from the existing DigiDoc file might not be copied to the new container.

Release the respective interface pointers after the end of working with ComSignedDoc object:

```
g_pSigDoc->Release();
g_pLib->Release();
```

The last task is to shut down the library and free the resources by calling the following function:

```
CoUninitialize();
```

Examples given in the previous sections described creating a new DigiDoc document from scratch. When working with an existing DigiDoc file then you can open and read the file as follows (the ComSignedDoc has to be created and initialized beforehand):

```
BSTR fname = _com_util::ConvertStringToBSTR("<input ddoc file's name>");

hr = g_pSigDoc->readSignedDoc(fname, // file name and path
      0,      // specifies whether file's hash is checked (1) or not (0)
      0,      // cache size of data files
      &lErr); // return code
```

The second parameter of the function above is a flag indicating whether checking hash value(s) of data file(s) is required at the time of opening. The third parameter can be used to specify the maximum size of ComDataFile content to be cached in memory.

After reading in an existing DigiDoc document, you can add signatures, data files and OCSP confirmations to it in the same way as described in the previous sections.

3.4.7. Verify signatures and OCSP confirmations

Note: in order to verify files that contain special characters, the system's regional settings have to be set to UTF-8 encoding.

You can verify all signatures and OCSP confirmations of a DigiDoc document which has previously been read in (i.e. the ComSignedDoc object has also been created and initialized) at once by using function:

```
IComSignedDoc.verifySigDoc(BSTR szDataFile, long *nRetResult);
```

For verifying a single signature of a DigiDoc document, do:

```
HRESULT hr = 0;
long l = 0;
int nIdx; // signature's index
int lErr = 0;
IComSignatureInfo* pSig = NULL;
// create ComSignatureInfo object
hr = CLSIDFromProgID(OLESTR("DigiDocLibCOM.ComSignatureInfo"), &g_clsid);
hr = CoCreateInstance(g_clsid, NULL, CLSCTX_INPROC_SERVER,
      __uuidof(IComSignatureInfo), (LPVOID *)&pSig);

hr = g_pSigDoc->getSignature(nIdx, pSig); // find the signature object by
             // its index in DigiDoc container (counting from zero)

// verify the signature
hr = g_pSigDoc->verifySignatureInfo(pSig, NULL, &lErr);

pSig->Release();
```

If you want to verify all of the DigiDoc file's signatures one by one then firstly find the number of existing signatures of the file with function IComSignedDoc.get_nSignatures(). Next, call the functions that are shown above for each signature, i.e. iterate over the amount of signatures by altering the nIdx value according to the number of existing signatures.

4. DigiDoc COM utility program

DigiDoc COM library includes a command line utility program – digidoccom.exe – which can be used to read and digitally sign files in OpenXadES format and verify signatures. The program has been implemented in C++ and built with MS Visual Studio 2008. Source code of the program is in file digidoccom.cpp.

The general format is:

```
> digidoccom [command(s)]
```

A list of all the available commands and their format can always be displayed by using the -? or –help commands:

```
> digidoccom -help
```

Note: DigiDoc COM library (including the utility program) retrieves configuration data from CDigiDoc library's configuration file 'digidoc.ini'.

4.1. General commands

- **-? or –help** – displays help about command syntax.
- **-in <input-digidoc-file> -** reads in a DigiDoc file
- **-new –** creates a new DigiDoc container
- **-add <input-file> <mime-type> –** adds a data file to a DigiDoc container
- **-out <output-file> –** creates a DigiDoc file at the specified location
- **-extract <data-file-id> <output-file> –** extracts DigiDoc file's contents

Creating new DigiDoc files

-new [format] [version]

Creates a new DigiDoc container with the specified format and version. The command allows creating DigiDoc containers with the currently default ddoc format and version, i.e. **DIGIDOC-XML 1.3**. If the command's argument values are left unspecified then the default format and version are chosen automatically.

-add <input-file> <mime-type> [content-type]

Adds a new data file to an unsigned DigiDoc container. If the container doesn't exist then creates one in the default format and version, i.e. DIGIDOC-XML 1.3.

Input file (required) specifies the name of the data file (it is recommended to include full path in this parameter; the path is removed when writing to DigiDoc container file).

Mime type (required) represents the MIME type of the original file like "text/plain" or "application/msword".

Content type (optional) reflects how the original files are embedded in the container. EMBEDDED_BASE64 option is supported and used by default.

-out <output-file>

Stores the newly created or modified DigiDoc document in a file.

Reading and extracting DigiDoc file's contents

-in <input-digidoc-file>

Specifies the input DigiDoc file name. It is recommended to pass the full path of the DigiDoc file in this parameter.

-extract <data-file-id> <output-file>

Extracts the selected data file from the DigiDoc container and stores it in a file.

Data file id (required) represents the ID for data file to be extracted from inside the DigiDoc container (e.g. D0, D1…).

Output file (required) represents the name of the output file.

Sample commands for creating, modifying and extracting DigiDoc files:

```
Sample: creating new DigiDoc file without signing

> digidoccom -new -add c:\temp\test1.txt text/plain -out c:\temp\test1.ddoc

  Input:
  - c:\temp\test1.txt   - a data file to be added to container
  - text/plain          - mime type of the data file
  - c:\temp\test1.ddoc  - container to be created


Sample: Adding multiple data files to an existing unsigned DigiDoc container
> digidoccom -in c:\temp\test1.ddoc -add C:\temp\test2.txt text/plain -add
C:\temp\test3.txt text/plain -out c:\temp\test1.ddoc

Input:
  - c:\temp\test1.ddoc          - unsigned container to be read and modified
  - C:\temp\test2.txt           - first data file to be added
  - C:\temp\test3.txt           - second data file to be added
  - text/plain                  - mime type of the data files
  - c:\temp\test1.ddoc          - output (modified) digidoc container


Sample: Extracting a data file from an existing DigiDoc file
> digidoccom -in c:\temp\test1.ddoc -extract D0 c:\temp\test1_ext.txt

Input:
- c:\temp\test1.ddoc    - the digidoc file to be extracted from
- D0                    - the data file ID to be extracted
- c:\temp\test1_ext.txt - file for storing the extracted data
```

4.2. Digital signature commands

- **-sign –** signs a DigiDoc file
- **-list –** displays a DigiDoc file's content info and verifies signature(s)
- **-verify –** displays and verifies DigiDoc file's signature(s)

Signing DigiDoc files

Signing is done by using the CSP/CAPI implementation. In the course of the signing operation, the CSP driver opens a dialog box for entering the signer's PIN2 code.

-sign [[manifest] [[city] [state] [zip] [country]]]

Adds a digital signature to a DigiDoc document. Note that adding signatures to DigiDoc files in older formats SK-XML, DIGIDOC-XML 1.1 and DIGIDOC-XML 1.2 is no longer supported. You can use the command with the following optional parameters:

manifest	Role or resolution of the signer
city	City where the signature is created
state	State or province where the signature is created
zip	Postal code of the place where the signature is created
country	Country of origin. ISO 3166-type 2-character country codes are used (e.g. EE)

Sample commands for signing DigiDoc files:

```
Sample: creating new DigiDoc file with signing

> digidoccom -new -add c:\temp\test2.txt text/plain -sign -out
c:\temp\test2_s.ddoc

Input:
  - c:\temp\test2.txt   - a data file to be added to container
  - text/plain          - mime type of the data file
  - c:\temp\test2_s.ddoc       - container to be created

Sample: signing an existing DigiDoc container (adding signatures)
> digidoccom -in c:\temp\test1.ddoc -sign -out c:\temp\test1_s.ddoc

Input:
  - c:\temp\test1.ddoc         - container to be signed
  - c:\temp\test1_s.ddoc       - output (modified) digidoc container
```

Listing DigiDoc file's contents and verifying signatures

Note: in order to verify files that contain special characters, the system's regional settings have to be set to UTF-8 encoding.

-list

Displays the data file and signature info of a DigiDoc document just read in; verifies all signatures.

Note: return code '0' refers to successful verification.

Returns:

- **Digidoc container data**, in format:

  SignedDoc | <format-identifier> | <version> | <verification return code for all of the document's signatures>

  For example: SignedDoc | DIGIDOC-XML | 1.3 | 0

- **List of all data files**, in format:

  DataFile | <file identifier> | <file name> | <file size in bytes> | <mime type> | <data file embedding option>

  For example: DataFile | D0 | test1.txt | 44 | text/plain | EMBEDDED_BASE64

- **List of all signatures** (if existing), in format:

  Signature | <signature identifier> | <data of the signer's certificate's 'Subject' field> | <verification return code> |

  For example: Signature | S0 | C=EE / O=ESTEID / OU=digital signature / CN=MÄNNIK,MARI-LIIS,47101010033 / SN=MÄNNIK / GN=MARI-LIIS / serialNumber=47101010033 | 0 |
- **Signer's certificate's information** (if signature(s) exist)
- **Revocation values**
- **OCSP responder certificate's information**

-verify

Analogous to **-list** command.

Sample commands for listing DigiDoc file's contents and verifying signatures:

```
Sample: listing DigiDoc file's contents, not signed
> digidoccom -in c:\temp\test1.ddoc -list

Input:
- c:\temp\test1.ddoc – the digidoc file which contents are to be listed

Returns:
SignedDoc|DIGIDOC-XML|1.3|0
DataFile|D0|test1.txt|54|text/plain|EMBEDDED_BASE64
DataFile|D1|test2.txt|19|text/plain|EMBEDDED_BASE64
DataFile|D2|test3.txt|52|text/plain|EMBEDDED_BASE64

Sample: listing DigiDoc file's contents and verifying signatures, signed
> digidoccom -in c:\Temp\test2_s.ddoc -verify

Input:
- C:\temp\test2_s.ddoc  – the digidoc file to be verified

Returns:
SignedDoc|DIGIDOC-XML|1.3|0
DataFile|D0|test2.txt|19|text/plain|EMBEDDED_BASE64
Signature|S0|C=EE/O=ESTEID/OU=digital signature/CN=M—NNIK,MARI-
LIIS,47101010033/SN=M—NNIK/GN=MARI-LIIS/serialNumber=47101010033|0|
X509Certificate|110860821818752731086954398181911413933|C=EE/O=AS
Sertifitseerimiskeskus/CN=TEST of ESTEID-SK
2011/emailAddress=pki@sk.ee|C=EE/O=ESTEID/OU=digital
signature/CN=M—NNIK,MARI-LIIS,47101010033/SN=M—NNIK/GN=MARI-
LIIS/serialNumber=47101010033|2011-08-31T13:14:57Z|2016-08-21T20:59:59Z
RevocationValues|N0|C=EE/O=AS Sertifitseerimiskeskus/OU=OCSP/CN=TEST of SK
OCSPRESPONDER 2011/emailAddress=pki@sk.ee|2012-05-11T08:02:32Z
X509Certificate|138983222239407220571566848351990841243|C=EE/O=AS
Sertifitseerimiskeskus/CN=TEST of EE Certification Centre Root
CA/emailAddress=pki@sk.ee|C=EE/O=AS Sertifitseerimiskeskus/OU=OCSP/CN=TEST
of SK OCSP RESPONDER 2011/emailAddress=pki@sk.ee|2011-03-07T13:22:45Z|2024-
09-07T12:22:45Z
```

Appendix 1: DigiDoc COM classes

1. ComSignedDoc

Represents the whole DigiDoc document, contains all of the subelements. Enables reading, verifying and writing the DigiDoc document and accessing the document's subelements.

Attributes

Name	Description
szFormat	Format of the DigiDoc document. Constant DIGIDOC_XML_1_1_NAME.
szFormatVer	Version of the DigiDoc document. Constant SK_XML_1_3_VER.
nDataFiles	Number of data files. Long data type, not editable.
nSignatures	Number of signatures. Long data type, not editable.
nNotaries	Number of OCSP confirmations. Long data type, not editable.

Methods

Name	Description
initialize	Initializes new DigiDoc document. Calling this method should precede all other DigiDoc document related actions when creating a new document.
szFormat	Document's format. String data type. Constant DIGIDOC_XML_1_1_NAME
szVersion	Document's version. String data type. Constant DIGIDOC_XML_1_3_NAME
readSignedDoc	Reads an existing DigiDoc document from a file. Returns an error code. szFileName- file name, String data type checkFileDigest – specifies if the file's hash is checked (1) or not (0) nMaxDFLen – size of the data file's cache
createSignedDoc	Saves the whole DigiDoc document to a file. Returns an error code. szOutputFile – name of the output file. String data type. szOldFileName – name of the existing file. String data type. To be used when an existing DigiDoc file is read in, possibly modified and written to a file again. Empty string in case of working with a new DigiDoc file.

Name	Description
createDataFile	Adds a new data file object to DigiDoc document and returns its reference. szFileName – file name. String data type. szContentType - specifies how data file is embedded in the DigiDoc container. Value CONTENT_EMBEDDED_BASE64 should be used – file is embedded in binary base64-encoded format. szMimeType - MIME type of the data file. For example "text/plain", "application/msword" or "application/pdf", depending on the data file's type. String data type. nSize - size of the data file in bytes. Long data type. If value is set to 0 then method calculateDataFileSizeAndDigest() has to be called later to let the library determine the value itself. baDigest - hash of the data file. Byte vector data type. If value is left undetermined (empty-Variant) then method calculateDataFileSizeAndDigest() has to be called later to let the library determine the value itself. nDigLen - size of the hash of the data file. Constant DIGEST_LEN. If value is set to 0 then library determines the value itself. szDigType - type of hash algorithm (only SHA-1 is supported). Constant DIGEST_SHA1_NAME. szCharset - data file's charset. UTF-8 encoding should be used. Constant CHARSET_UTF_8.
getDataFile	Returns data file's object based on the provided index value. nIndex – data file's index. Long data type. pStoreTo - ComDataFile object for returning the result.
getDataFileWithId	Returns data file's object based on the provided data file identifier value. szId – data file's identifier. String data type. pStoreTo - ComDataFile object for returning the result.
calculateDataFileSizeAndDigest	Calculates data file's size and hash so that the values don't have to be specified when calling out function createdDataFile(). szId – data file's identifier. String data type. szFileName - data file's name. digType – hash type (only SHA-1 is supported). Constant DIGEST_SHA1_NAME.
createSignatureInfo	Creates a new empty signature to a DigiDoc document and returns the signature object. pStoreTo - ComSignatureInfo object for returning the result.

Name	Description
getSignature	Returns signature object based on the index value provided. nIndex – signature's index. Long data type. pStoreTo - ComSignatureInfo object for returning the result.
getSignatureWithId	Returns signature object based on the signature's identifier value provided. szId – signature's identifier. String data type. pStoreTo - ComSignatureInfo object for returning the result.
addAllDocInfos	Adds all of the current DigiDoc document's data files to the signature object. pSigInfo - ComSignatureInfo object for returning the result.
getNotaryInfo	Returns OCSP confirmation object (ComNotaryInfo) based on the index value provided. nIndex – confirmation object's index. Long data type. pStoreTo - ComNotaryInfo object for returning the result.
getNotaryWithId	Returns OCSP confirmation object (ComNotaryInfo) based on the identifier value provided. szId – identifier. String data type. pStoreTo - ComNotaryInfo object for returning the result.
getNotaryWithSigId	Returns OCSP confirmation object (ComNotaryInfo) based on the signature's identifier value provided. szSigId – signature's identifier. String data type. pStoreTo - ComNotaryInfo object for returning the result.
createNotaryInfo	Creates a new ComNotaryInfo object. pSigInfo – signature which is going to be confirmed. pStoreTo - ComNotaryInfo for storing the confirmation.
createNotaryInfoFromFile	Adds a new confirmation from a file. pSigInfo – signature to be verified. szOcspRespFile - OCSP response file's name. szNotaryCertFile – OCSP responder's certificate file. pRetError – variable for saving the returned error code. Long data type. pStoreTo - ComNotaryInfo object for storing the new OCSP confirmation data.
verifyDigest	Checks the DigiDoc document's hash value. Returns error code. pSigInfo - ComSignatureInfo object. pDocInfo - ComDocInfo object. szFileName – file name. String data type.

Name	Description
verifyMimeDigest	Checks the mime type’s hash value. Returns error code. pSigInfo - ComSignatureInfo object. pDocInfo - ComDocInfo object. szFileName - file name. String data type.
verifySignatureInfo	Verifies the signature. Returns error code. Certificates are read from file. pSigInfo - ComSignatureInfo object. szDataFile – file name. String data type, may be left unspecified. pComFilePathArray - ComFilePathArray object.
verifySignatureInfoCERT	Verifies the signature. Returns error code. Certificates are found based on search criterion. pSigInfo - ComSignatureInfo object to be verified pSignerCACert - ComCertSearch object for searching CA certificate. szDataFile – file name. String data type, may be left unspecified.
verifyNotaryInfo	Verifies the OCSP confirmation. Returns error code. Certificates are read from file. pNotInfo - ComNotaryInfo object.
verifyNotaryInfoCERT	Verifies the OCSP confirmation. Returns error code. Certificates are found based on search criterion. pNotaryInfo - ComNotaryInfo object to be verified certJuurSK - ComCertSearch object for searching the CA root certificate. certEstEidSK - ComCertSearch object for searching the EstEID CA certificate. szCApath - CA certificates’ directory path. String data type. notCert - ComCertSearch object for searching the OCSP responder certificate.
verifySigDoc	Verifies the while DigiDoc document, including its signatures and OCSP confirmations. Returns error code. Certificates are read form file. szDataFile – file name. String data type, may be left unspecified. pComFilePathArray - ComFilePathArray object.

Name	Description
verifySigDocCERT	Verifies the while DigiDoc document, including its signatures and OCSP confirmations. Returns error code. Certificates are found based on search criterion. signerCA - ComCertSearch object for searching the signer's CA certificate. notaryCA - ComCertSearch object for searching the CA OCSP responder's certificate. rootCA - ComCertSearch object for searching the CA root certificate. szCAPath - CA certificate's directory path. String data type. notCert – ComCertSearch object for searching the OCSP responder's certificate. szDataFile – data file name. String data type, may be left unspecified. pComFilePathArray - ComFilePathArray object.
deleteDataFile	Removes a data file. Returns error code. szDataFileId – data file's identifier. String data type.
deleteNotaryInfo	Removes an OCSP confirmation. Returns error code. szNotInfoId – OCSP confirmation's identifier. String data type.
deleteSignatureInfo	Removes a signature. Returns error code. szSigInfoId – signature's identifier. String data type.
getConfirmation	Sends an OCSP confirmation request and checks the response. Returns error code. pSigInfo - ComSignatureInfo object. pkcs12File – OCSP responder access certificate's PKCS12 file name. String data type. password - OCSP responder access certificate's PKCS12 container's password. String data type. notaryURL - OCSP responder's URL. String data type. proxyHost – proxy URL. String data type. proxyPort – proxy port. String data type.
getFilePathArray	Initializes vector of document's files. Returns the number of files as a vector. pFilePathArray - ComFilePathArray objekt.
getDataFileNameWidthId	Returns data file's name based on its identifier. Id – file's identifier. String data type.

2. *ComDataFile*

Represents a data file of a DigiDoc document.

Attributes

Name	Description
szId	Data file's identifier. String data type. Not editable.
szFileName	Data file's name. String data type. Not editable.
szMimeType	Data file's mime type. String data type. Not editable.
szContentType	File embedding mode. String data type. Not editable.
nSize	File size. Long data type. Not editable.
szDigestType	Hash type. String data type. Not editable.
baDigest	Hash value. Byte vector. Not editable.
nDigestLen	Hash size. Long data type. Not editable.
nAttributes	Number of attributes. Long data type. Not editable.
szaAttNames	Attribute names. String vector. Not editable.
szaAttValues	Attribute values. Not editable.

Methods

Name	Description
addAttribute	Adds an attribute. szAttrName – attribute name. String data type. szAttrValue – attribute value. String data type.
getAttributeName	Returns attribute's name. nIndex – attribute's index. Long data type.
getAttributeValue	Returns attribute's value. nIndex – attribute's index. Long data type.

3. ComDocInfo

Signed attributes of one particular data file.

Attributes

Name	Description
szDocId	Data file's identifier. String data type.
szDigestType	Hash type. Constant DIGEST_SHA1_NAME.
nDigestLen	Hash size. Long data type, not editable.
baDigest	Hash value. Byte vector.
nMimeDigestLen	Mime type's hash size. Long data type, not editable.
baMimeDigest	Mime type's hash value. Byte vector.

4. ComSignatureInfo

Holds the data of a signature.

Attributes

Name	Description
szId	Unique identifier. String data type, not editable.
nDocs	Number of documents. Long data type, not editable.
szTimeStamp	Time of signature creation. Should be retrieved by using the ComTimeStamp.getString() method. String data type.
nSigPropDigestLen	Size of the signed parameters' hash. Long data type, not editable.
szSigType	Signature's type. String data type.
nSigLen	Signature's size. Long data type, not editable.
nIssuerSerial	Certificate's number. Long data type.
nCertDigestLen	Size of the certificate's hash. Long data type, not editable,

Methods

Name	Description
verifySigPropDigest	Necessary if file might be created with a different library, i.e. file's format might not be predictable. Returns error code or ERR_OK in case of success.
	pSigDoc – ComSignedDoc object, needed for verification. szDataFile – data file.
addDocInfo	Adds a ComDocInfo object to signature. szDocId – document's unique identifier. String data type. szDigType – hash type. Constant DIGEST_SHA1_NAME. baDigest – hash value. Byte vector. nDigLen – hash size. Constant DIGEST_LEN. baMimeDig - mime type's hash value. Byte vector. nMimeDigLen – mime type's hash size. Constant DIGEST_LEN. pStoreTo – ComDocInfo for storing the created object.
calculateSignatureWithEstID	Calculates signature with EstEID smart card. Returns error code. pSigDoc - ComSignedDoc object. nSlot – smart card's signature slot. szPasswd – PIN code for signing.

Name	Description
calculateSignatureWithCSPEstID	Calculates. Signature with EstEID smart card by using CSP. Returns error code. pSigDoc - ComSignedDoc object.
addSignerRole	Adds signer's role. nCertified – specifies if the role is certified (1) or not (0). Long data type. szRole – role name. String data type. nRoleLen – length of the role name. Long data type. nEncode – specifies if the role should be encoded (1) or not (0). Long data type.
countSignerRoles	Returns the number of signer's roles. nCertified – specifies if certified (1) or not certified (0) roles are counted. Long data type.
getSignerRole	Returns the name of the signer's role. nCertified - specifies if certified (1) or not certified (0) roles are counted. Long data type. nIndex – role's index. Long data type.
createXMLSignedProperties	Creates "<SignedProperties>" XML block and returns it. pSigDoc - ComSignedDoc object.
setSignatureCertFile	Determines the signature's certificate and calculates hash. Returns error code. szCertFile – certificate file. String data type.
calculateSignedPropertiesDigest	Calculates <SignedProperties> element's hash value. Returns error code. pSigDoc - ComSignedDoc object.
getSignCertData	Returns signature's certificate as a long data type.
getDocInfo	Returns ComDocInfo object based on index value provided. nIndex - DocInfo index. Long data type. pStoreTo - ComDocInfo object for storing the result.
getDocInfoWithId	Returns ComDocInfo object based on identifier value provided. szId - DocInfo unique identifier. String data type. pStoreTo - ComDocInfo object for storing the result.

Name	Description
calculateSigInfoSignature	Calculates signature value. Returns error code. pSigDoc - ComSignedDoc object. nSigType – signature type. Constant DIGEST_SHA1 szKeyFile - RSA key's file. String data type. szPasswd – key's password. String data type. szCertfile – certificate file. String data type.
verifySigCert	Verifies the signed attributes of signature. Returns error code.
setSignatureProductionPlace	Determines signature creation place. Missing or unimportant values may be left unspecified. szCity – City, string data type. szState – State or county, string data type. szZipCode – Zip code, string data type. szCountry - Country, string data type. szCharset – charset of the data. Constant CHARSET_UTF_8.
getSignatureProductionPlace	Returns the signature creation location information. Missing parameters are returned as empty strings. pszCity - City, string data type. pszState - State or county, string data type. pszZipCode - Zip code, string data type. pszCountry - Country, string data type.

5. ComNotaryInfo

Holds a signature's OCSP confirmation data.

Attributes

Name	Description
szId	Unique identifier. String data type.
szSigId	Signature's unique identifier. String data type.
szNotType	OCSP protocol type. Constant SK_NOT_VERSION.
timeProduced	Creation time. In the format of ComTimestamp.
szRespIdType	OCSP responder's type identifier. String data type.
baRespId	OCSP responder's type. Byte vector, not editable.
nRespIdLen	OCSP responder's type size. Long data type, not editable.
thisUpdate	Last update. String data type.
nextUpdate	Next update. String data type.

Name	Description
nCertNr	Certificate's number. Long data type.
szDigestType	Hash type. Constant DIGEST_SHA1_NAME.
szSigType	Signature type. Constant SIGN_RSA_NAME
baSigValue	Signature value. Byte vector, not editable.
nSigLen	Signature's size. Constant SIGNATURE_LEN.
nIssuerSerial	Certificate's number Long data type.
baCertDigest	Certificate's hash value. Byte vector, not editable.
nCertDigestLen	Size of the certificate's hash. Constant DIGEST_LEN.
baOcspDigest	OCSP response's hash. Byte vector, not editable.
nOcspDigestLen	OCSP response hash size. Long data type, not editable.

Methods

Name	Description
verifyNotCert	Checks OCSP responder's certificate. Returns error code.
getNotCertData	Returns OCSP responder's certificate.
verifyNotaryDigest	Checks OCSP responder's hash value. Returns error code. pUseFormatOf - ComSignedDoc object whose format's version should be used.

6. ComDigiDocLib

Contains general methods which are not bound to any particular class.

Methods

Name	Description
getLibName	Returns the library's name as a String value.
getLibVersion	Returns the library's version as a String value.
getSupportedFormats	Returns the formats that are supported by the library as a String vector.
extractDataFile	Extracts a data file from DigiDoc document and stores it to the specified file. pSignedDoc - ComSignedDoc object. szFileName – document's name. String data type. szDataFileName – data file's name. String data type. szDocId – data file's identifier. String data type. szCharset – charset of the output data. Constant CHARSET_UTF_8.

Name	Description
getCertSubjectName	Determines certificate owner's name in the specified format. Returns error code. hCertHandle – certificate handle. Long data type. pCopyTo – String variable for storing the result.
getCertIssuerName	Determines the certificate's issuer name in the specified format. Returns error code. hCert – certificate handle. Long data type. pszStoreTo - String variable for storing the result.
getCertSerialNumber	Determines the certificate's serial number. Returns error code. hCert – certificate handle. Long data type. pszStoreTo - String variable for storing the result.
getCertNotBefore	Determines the valid from date of the certificate in ComTimeStamp format. Returns error code. pUseFormatOf - ComSignedDoc object whose format version is to be used. hCert – certificate handle. Long data type. szTimeStamp - String variable for storing the date value.
getCertNotAfter	Determines the valid to date of the certificate in ComTimeStamp format. Returns error code. pUseFormatOf - ComSignedDoc object whose format version is to be used. hCert – certificate handle. Long data type. szTimeStamp - String variable for storing the date value.
saveCert	Saves the certificate file. Returns error code. hCert – certificate handle. Long data type. szSaveToFile – name of the output file. String data type. nFormat – file format. Constant FILE_FORMAT_ASN1 or FILE_FORMAT_PEM.
isCertValid	Checks the validity of the certificate. Returns error code. hCert – certificate handle. Long data type. TimeValue – specified time value.
isCertSignedBy	Checks if the certificate has been signed by the respective CA. Returns error code. hCert – certificate handle. Long data type. szCAFile – CA certificate file. String data type.

Name	Description
isCertSignedByCERT	Checks the certificate has been signed with the respective certificate. Returns error code. searchCert - ComCertSearch object for searching the certificate to be checked. searchCACert - ComCertSearch object for searching the CA certificate.
readCertificateByPKCS12OnlyCert	Returns certificate handle, can be used to read OCSP responder access certificate from PKCS#12 container pkcs12File – PKCS12 file name. String data type. password - PKCS12 container’s password value. String data type.

7. ComErrorInfo

Enables error handling. Offers functionality for accessing the library’s error buffer and holds the data of errors that have been read.

Error handling offered by the library has been combined in two levels:

1. If a method returns an error code of long data type then the error code should be stored. Then, explanation of the error code should be retrieved in text format by using method ComErrorInfo.getErrorStringByCode().
2. If the method doesn’t return error code then ComErrorInfo.getNextUnreadError() method should be called. The method returns ‘true’ if there are any unread errors in the library. The first unread error is going to be stored in the ComErrorInfo object.

Generally, it is useful to create a unitary error handling functionality in client application which checks the given error code and then iterates through buffer of unread errors.

Note: error handling functionality uses the library’s global variables, i.e. the class is not thread-safe!

Attributes

Name	Description
nCode	Error code. Long data type, not editable.
szFileName	Name of the source code file. String data type, not editable.
nFileLine	Line number in the source code file. Long data type, not editable.
szAssertion	Explanation. String data type, not editable.

Methods

Name	Description
hasContent	If an error has been stored in the object (i.e. the attributes have values) then returns ‘true’ as Variant type.
getNextUnreadError	Reads in the next error. If the buffer of unread errors is empty then returns ‘false’ with Variant data type.

Name	Description
getErrorString	Returns the textual representation of the error that has been read in.
clearErrorInfo	Clears the information of currently read in error.
clearAllUnreadErrors	Clears the whole buffer of unread errors.
getErrorStringByCode	Returns error text based on error code. nCode – error code. Long data type.
getErrorClass	Returns error class based on error code. nCode – error code. Long data type. Possible values for error class are: • 0 – no errors • 1 – technical error • 2 – user's error • 3 – library's error

8. *ComTimestamp*

Represents the date and time format that is used in DigiDoc documents. Holds and converts data of date and time of a DigiDoc document according to format determined by specification.

Attributes

Name	Description
year	Year, long data type.
month	Month, long data type.
day	Day, long data type.
hour	Hour, long data type.
minute	Minute, long data type.
second	Second, long data type.
tz	Time zone, long data type.

Methods

Name	Description
readFromString	Initializes the classes variables from a String type variable. pUseFormatOf - ComSignedDoc object whose format version is used. szReadFrom – time value to be read. String data type.
getString	Returns date in a String type variable according to the classes variables. pUseFormatOf - ComSignedDoc object whose format version is used.

9. ComCertSearchStore

Subclass for certificates' search from MS certificate store. The class should only be used via ComCertSearch class.

Attributes

Name	Description
szStoreName	Name of the certificate store. String data type.
nCertSerial	Certificate's serial number. Long data type.
nSubDNCriterias	Number of criteria which determine the subject's name. Long data type, not editable.
saSubDNCriterias	Criteria of the subject's name. String vector.
nIssuerDNCriterias	Number of criteria of the certificate's issuer. Long data type, not editable.
saIssuerDNCriterias	Criteria of the certificate's issuer. String vector.

Methods

Name	Description
addSearchType	Adds a search criterion. nCertSearchStoreType – one of the search criterion's constants CERT_STORE_SEARCH_BY_xxx.
removeSearchType	Removes a search criterion. nCertSearchStoreType - one of the search criterion's constants CERT_STORE_SEARCH_BY_xxx.

10. ComCertSearch

Common class for searching certificates and forwarding them to methods of other classes.

Attributes

Name	Description
nCertSearchBy	Determines the search type. Possible values: • CERT_SEARCH_BY_STORE, • CERT_SEARCH_BY_X509 • CERT_SEARCH_BY_PKCS12 – can be used to search for the OCSP responder access certificate's PKCS#12 file
szX509FileName	Name of the certificate file. String data type.
szKeyFileName	Name of the private key file. String data type.
pswd	PKCS#12 password.

szPkcs12FileName	PKCS#12 container's name. String data type.

Methods

Name	Description
getCertSearchStore	Returns ComCertSearchStore object. If the object doesn't exist then a new object is created. pStoreTo - ComCertSearchStore object for storing the result.
lookup	Certificate's search. If certificate is found then returns 'true', else 'false'.

11. ComCertificate

A class which holds and intermediates data of a certificate.

Attributes

Name	Description
hHandle	Handle which represents a concrete certificate. Can be used when calling out methods which use certificates. Long data type, not editable.
containsCert	Checks if object represents one concrete certificate or not. Variant/bool data type, not editable.
szPropSubjectName	Name of the certificate's subject. Otherwise equivalent to ComDigiDocLib.getCertSubjectName() method, except that error handling is not implemented – thus, should not be trusted. Exists only in the purpose of convenience. String data type, not editable.
szPropIssuerName	Name of the certificate's issuer. Otherwise equivalent to ComDigiDocLib.getCertIssuerName() method, except that error handling is not implemented – thus, should not be trusted. Exists only in the purpose of convenience. String data type, not editable.
nPropSerialNumber	Certificate's serial number. Otherwise equivalent to ComDigiDocLib.getCertSerialNumber() method, except that error handling is not implemented – thus, should not be trusted. Exists only in the purpose of convenience. String data type, not editable.

12. ComCertArray

Vector of ComCertificate objects. Enables searching from MS certificate store if the search returns more than one certificate.

Attributes

Name	Description

nCerts	Number of certificates in vector.

Methods

Name	Description
getCertByIndex	Determines the ComCertificate object in the vector with the provided index value. nIndex – index in the vector. Allowed values are in the range of [0..nCerts-1]. Long data type. pStoreTo - ComCertificate object for storing the certificate.
clear	Clears the whole vector.
searchBySearchStore	Searches for certificates from MS certificate store. Stores the search results to vector. Returns error code. pSearchStore - ComCertSearchStore object which determines the search criteria.

13. ComPolicyIdentifierArray

Vector which holds the rules of certificate's usage.

Attributes

Name	Description
m_nPolicyIdentifierCount	Number of rules for certificate's usage.

Methods

Name	Description
read_CertPolicies	Reads the usage rules of the specified certificate. Returns the number of usage rules in vector (the same value as in attribute nPolicyIdentifierCount). X509 – reference to the certificate's X509 structure.
get_szCPS	nIndex – index in vector. Allowed values are in the range of [0..m_nPolicyIdentifierCount-1]. Long data type.
get_szOID	nIndex – index in vector. Allowed values are in the range of [0..m_nPolicyIdentifierCount-1]. Long data type.
get_szUserNotice	nIndex - index in vector. Allowed values are in the range of [0..m_nPolicyIdentifierCount-1]. Long data type.
get_isCompanyPolicy	nIndex - index in vector. Allowed values are in the range of [0..m_nPolicyIdentifierCount-1]. Long data type. Returns long-type value 1 if true, otherwise 0.

14. *ComFilePathArray*

Vector of file names.

<u>**Attributes**</u>

Name	Description
m_nFilePathCount	Number of files on the vector.

<u>**Methods**</u>

Name	Description
get_FileName	Reads the file name (without path). nIndex – index in vector. Allowed values are in the range of [0.. nFilePathCount -1]. Long data type.
get_FilePath	Reads the file name with full path value. nIndex – index in vector. Allowed values are in the range of [0.. nFilePathCount -1]. Long data type.
set_FilePath	Saves the file name with full path value. nIndex – index in vector. Allowed values are in the range of [0.. nFilePathCount -1]. Long data type.